1999-12

Resolving frequency ambiguities in step : frequency compressive receivers

Ceylan, Oktay.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

RESOLVING FREQUENCY AMBIGUITIES IN STEP-FREQUENY COMPRESSIVE RECEIVERS

by

Oktay Ceylan

December 1999

Thesis Advisor: D. Curtis Schleher
Second Reader: David Jenn

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1999	Master's Thesis

4. TITLE AND SUBTITLE
RESOLVING FREQUENCY AMBIGUITIES IN STEP-FREQUENCYCOMPRESSIVE RECEIVERS

5. FUNDING NUMBERS

6. AUTHOR(S)
Ceylan, Oktay

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

Present Compressive Receiver implementations are limited due to their analog implementation and the necessity for digital processing of the serial output data. Previous research has shown that a stepped-frequency digital design using sub-Nyquist sampling mitigates many of the limitations. An algorithm that implements the Chinese Remainder Theorem to solve the frequency ambiguities that occur in the design due to sub-Nyquist sampling with high resolutions is investigated. Different resolutions, a different number of sampling frequencies, and sampling frequency pairs and triples with various differences are simulated for one to five signals that overlap in the time domain. Predictions for the best achievable resolution, the minimum number of sampling frequencies needed, and the difference required between the sampling frequencies are made according to the comparison of simulation results.

14. Subject Terms
Wideband Digital Compressive Receivers, Resolving Frequency Ambiguities, Chinese Remainder Theorem .

15. NUMBER OF PAGES
142

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18298-102

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

RESOLVING FREQUENCY AMBIGUITIES IN STEP-FREQUENCY COMPRESSIVE RECEIVERS

Oktay Ceylan
1st Lieutenant, Turkish Army
B.S., Turkish War Academy, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1999

Author: [redacted]
Oktay Ceylan

Approved by: [redacted]
D. Curtis Schleher, Thesis Advisor

[redacted]
David Jenn, Second Reader

[redacted]
Dan C. Boger, Chairman
Information Warfare Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Present Compressive Receiver implementations are limited due to their analog implementation and the necessity for digital processing of the serial output data. Previous research has shown that a stepped-frequency digital design using sub-Nyquist sampling mitigates many of the limitations. An algorithm that implements the Chinese Remainder Theorem to solve the frequency ambiguities that occur in the design due to sub-Nyquist sampling with high resolutions is investigated. Different resolutions, a different number of sampling frequencies, and sampling frequency pairs and triples with various differences are simulated for one to five signals that overlap in the time domain. Predictions for the best achievable resolution, the minimum number of sampling frequencies needed, and the difference required between the sampling frequencies are made according to the comparison of simulation results.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENT

I wish to express my gratitude to those who contributed to the successful and timely completion of this project.

First, I would like to thank Professor Philip Pace for being so generous and providing a place to study during the last nine months I spent at the Naval Postgraduate School. I also would like to thank Professor David Jenn for proofreading my thesis.

My deepest gratitude goes to Professor Curtis Schleher, whose guidance, patience, and cooperation are the pillars of this thesis.

Finally, I would like to thank my parents and my wife Arzu for their support and understanding during the development of this work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Although the theoretical advantages of Compressive Receivers have been recognized for many years, their practical implementation has been limited to experimental units in the ELINT frequency band (2 - 18 GHz), and operational units in the COMINT frequency band (2 - 500 MHz). This restricted use of Compressive Receivers is due to limitations in the areas of:

- The analog implementation
- The digital processing of the serial output data

Schleher has investigated the feasibility of a new type of wideband digital compressive receiver, which would overcome these difficulties [Ref. 1]. The feasibility of this receiver was demonstrated through simulation in 1997. It covers a bandwidth of 8 GHz, with 100 percent probability of intercept.

In the stepped-frequency compressive receiver approach, the restrictions due to the analog implementation and digital processing are overcome using sub-Nyquist sampling in combination with stepped-frequency chirp processing.

The stepped-frequency approach substantially reduces the bandwidth of the digital processing components from 32 GHz to 9 GHz. Unfortunately, this still requires an 18 GHz A/D converter for Nyquist rate sampling, which is an impractical number.

On the other hand, employing sub-Nyquist rate sampling reduces the bandwidth of the digital processing components to practical numbers, i.e., 1-2 GHz.

However, sub-Nyquist rate sampling has the undesirable effect of folding the frequency components of the sampled signal into the fundamental range between $\pm f_s/2$ and, consequently, causing ambiguities. Although there is no way to resove these ambiguities by using one set of data, they can be resolved by sampling the signal more than once.

This thesis develops an algorithm to solve the frequency ambiguities that occur in wideband digital compressive receiver design when we use sub-Nyquist sampling with high resolutions, such as 5, 10, or 25 MHz. This algorithm was coded in Matlab® and then some parameters were varied to see their effects.

We will start Chapter II by presenting some fundamentals of the Sampling and Chinese Remainder Theorems

and the General Product Rule. Chapter III will expand on the wideband digital compressive receiver design and the ambiguity problem. The development of the algorithm and the analysis of the results of its application are presented in Chapters IV and V, respectively. Chapter VI then summarizes and concludes this work and provides recommendations for future research. Matlab codes used in this thesis are included in Appendix A. The results of the experiments with different parameters are presented in Appendix B.

THIS PAGE INTENTIONALLY LEFT BLANK

II. FUNDAMENTALS

The objective of this chapter is to ensure that the reader is familiar with the basics of the Sampling and Chinese Remainder Theorems, and the General Product Rule. It does not intend to be a complete reference on these subjects, but to highlight those aspects of these subjects pertinent to this thesis.

A. SAMPLING THEORY

The sampling process may be studied in either the time-domain or frequency-domain. We use benefits of both approaches and focus on whichever one we find more helpful in understanding the important issues in sampling.

1. Sampling

Sinusoidal waveforms of the form

$$x(t) = A\cos(2\pi f_o t + \phi) \tag{2.1}$$

are examples of *continuous-time* signals. Continuous-time signals are represented mathematically by functions of time, $x(t)$, where t is a continuous variable.

A *discrete-time* signal is represented mathematically by an indexed sequence of numbers. We denote the values of such a sequence as $x[n]$, where n is the integer index

indicating the order of the values in the sequence. We can obtain discrete-time signals by sampling a continuous-time signal $x(t)$;i.e.,

$$x[n] = x(nT_s) \qquad -\infty < n < \infty$$

The operation of sampling is represented by a block diagram, as in Figure 2.1 [Ref. 2].

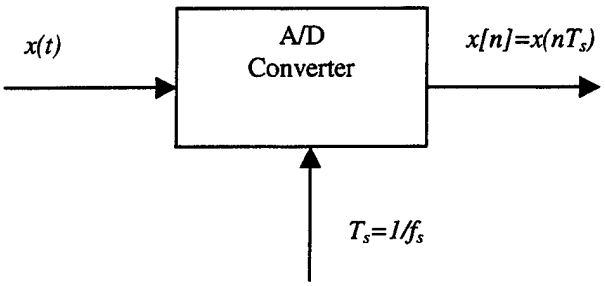

Figure 2.1 - Block Diagram Representation of Analog-to-Digital (A/D) Converter (After Ref. 2).

In order to understand the effect of sampling, consider an input band limited signal $x(t)$ that is multiplied by a sampling function $i(t)$ consisting of a series of impulses separated by the sample period $T_s=1/f_s$, then the sampled signal $x^*(t)$ will be:

$$x^*(t) = x(t) \cdot i(t)$$

$$x^*(t) = x(t) \cdot \sum_{-\infty}^{\infty} \delta(t - nT_s)$$

$$x^*(t) = \sum_{-\infty}^{\infty} x(nT_s) \cdot \delta(t - nT_s) \qquad (2.2)$$

where T_s = Sampling Period [Ref. 3].

Figure 2.2 shows this sampling process in the time and frequency domains. Notice how the sampling process changes the location of the various spectrum components that make up the signal. The spectrum of the sampled signal is a periodic replication of the unsampled signal spectrum separated by the sampling frequency [Ref. 4]. This is significant as it provides a means of understanding the effects of sampling on the signal.

2. Recovering the Original Signal and Aliasing

We can reconstruct the original band limited signal by applying an ideal bandpass filter to the sampled signal as long as we have enough sample points (Figure 2.3). The process of recovering original waveform from samples by filtering is illustrated in the frequency domain in Figure 2.4. These statements naturally raise the question of how frequently we must sample in order to retain enough information to reconstruct the original continuous-time signal. The answer to this question is given by the *Sampling Theorem,* which states that when the sampling rate is greater than twice the highest frequency component contained in the spectrum of the analog signal, the signal can be reconstructed from the samples. This minimum sampling rate is called the *Nyquist* rate.

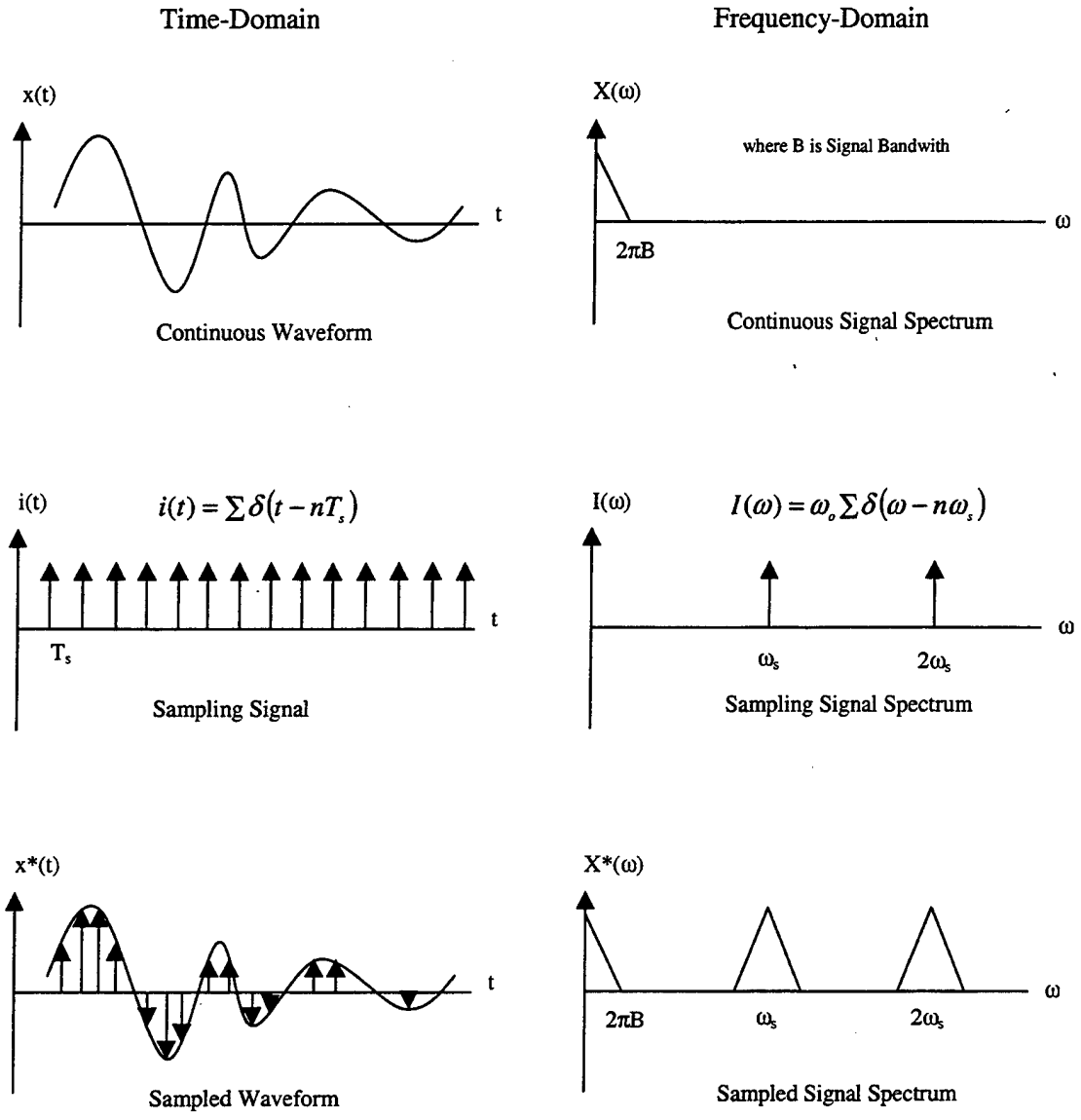

Figure 2.2 - Sampling in Time and Frequency Domains (After Ref. 3).

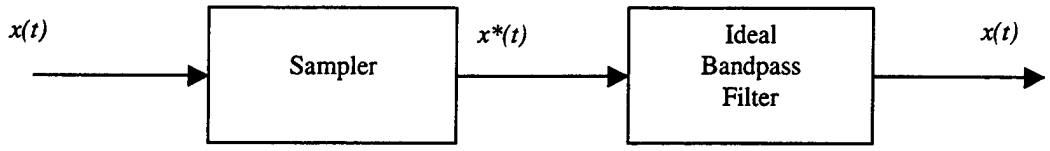

Figure 2.3 - Block Diagram Representation of Recovering Original Waveform (After Ref. 3).

Notice that the Nyquist rate implies that reconstruction of a signal is possible if we have at least two samples per period [Ref. 2]. Aliases will occur if we sample at a rate less than twice the highest frequency (Figure 2.5).

We can derive a formula that gives the inter-relationship between the signal frequency f_o, and the sampling rate f_s, by considering in greater detail the case of continuous-time sinusoid in Equation 2.1:

$$x[n] = x(nT_s) = A\cos(2\pi f_o nT_s + \phi) \qquad (2.3)$$

Now consider another sinusoid with the same amplitude and phase, but with a different frequency, f_o+mf_s, where m is an integer and $T_s=1/f_s$.

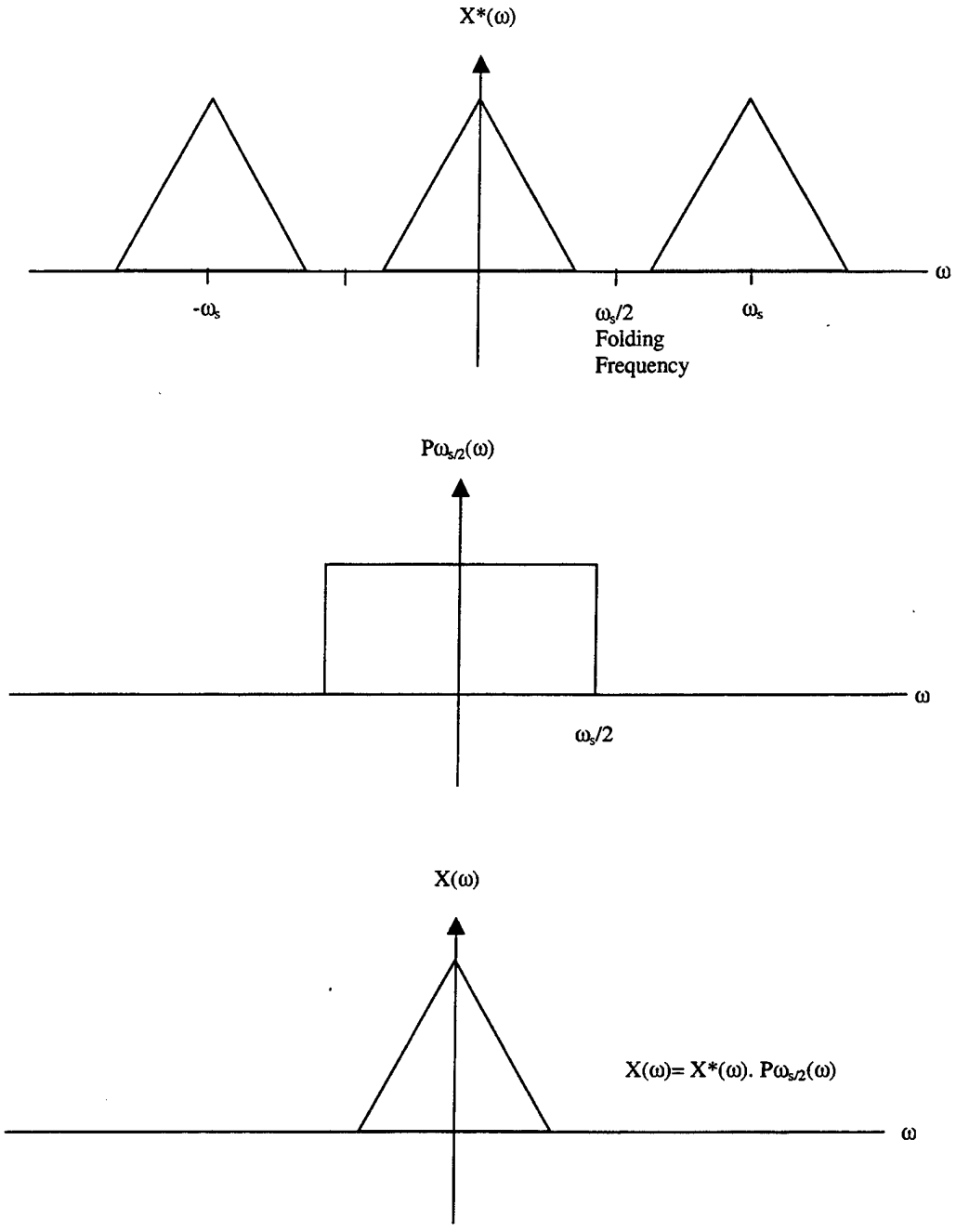

Figure 2.4 - The Process of Recovering Original Waveform from Samples by Filtering (After Ref. 3).

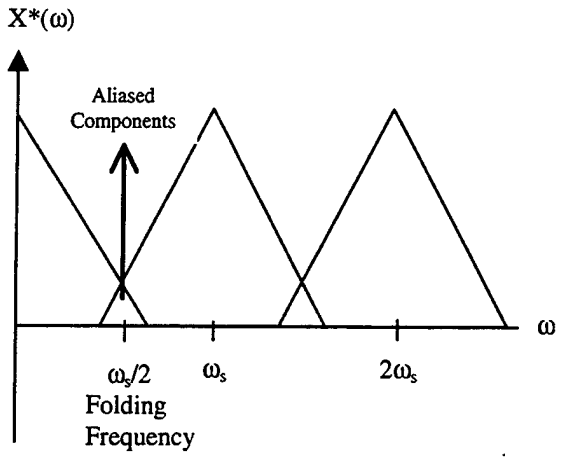

Figure 2.5 - Aliasing in Frequency Domain Due to Undersampling (After Ref .4).

If this second waveform, $y(t)$, is sampled with sampling frequency f_s, we get:

$$\begin{aligned} y[n] = y(nT_s) &= A\cos\left(2\pi\left(f_o + mf_s\right)nT_s + \phi\right) \\ &= A\cos\left(2\pi f_o nT_s + 2\pi mnf_s T_s + \phi\right) \\ &= A\cos\left(2\pi f_o nT_s + 2\pi mn + \phi\right) \\ &= A\cos\left(2\pi f_o nT_s + \phi\right) \\ &= x[n] \end{aligned} \qquad (2.4)$$

In other words, $y[n]$ has the same sample values as $x[n]$, so it is indistinguishable from $x[n]$. Since m was specified only as an integer (either positive or negative), this means that there are an infinite number of sinusoids that give the same sequence of samples as x[n]. These

infinite *aliases* of the frequency f_o with respect to the sampling frequency f_s will occur at frequencies $\pm f_o + mf_s$ [Ref 2].

3. Sub-Nyquist Sampling

To better understand the effect of sub-Nyquist sampling, consider the sampled signal $x^*(t)$ in Equation 2.2, which is the input waveform, $x(t)$, multiplied by the sampling function, $i(t)$. The spectrum of the sampled signal $x^*(t)$ equals:

$$X^*(\omega) = \sum_{-\infty}^{\infty} S(\omega + n\omega_s) \tag{2.5}$$

Note from Equation 2.5 that as n varies, signals produce frequency components in the frequency range between $\pm f_s/2$. Thus, sub-Nyquist sampling has the effect of translating the frequency components of the signal into the fundamental range between $\pm f_s/2$. The frequency translation of signals into the fundamental band by the use of sub-Nyquist sampling can be seen in Figure 2.6.

As stated earlier the digitization of a signal is usually governed by the Nyquist rate where the sampling frequency is at least twice the signal bandwidth. The

Nyquist rate however, places a limitation only on the information that can be derived from a single set of digitized data. If the sampling frequency is less than twice the bandwidth of the signal being digitized, aliasing, and consequently ambiguities occur. With additional information however, ambiguous frequency components due to undersampling may be resolved [Ref 5].

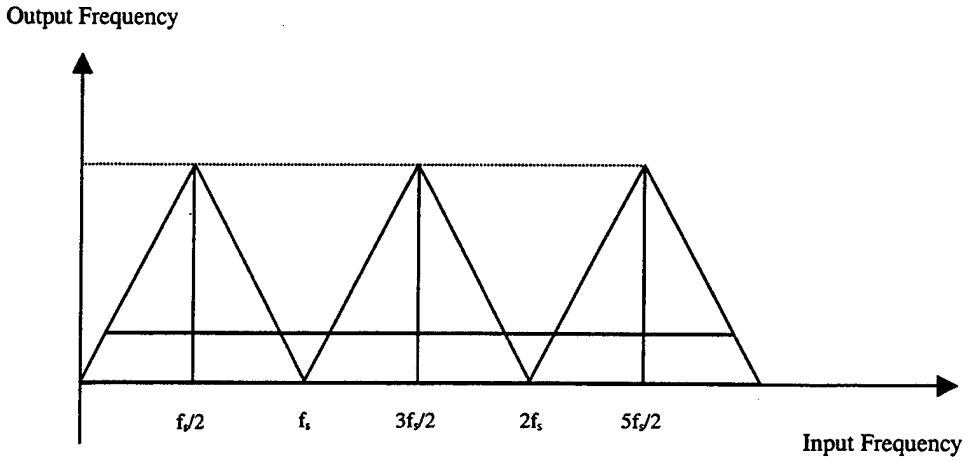

Figure 2.6 - Input Frequency Band versus Output Frequency Band for Sub-Nyquist Sampling (After Ref.1).

B. THE CHINESE REMAINDER THEOREM

Before introducing the Chinese Remainder Theorem, some definitions should be noted:

- Definition of *Relatively Prime*: The integers *a* and *b* are called relatively prime if *a* and *b* have a greatest common divisor of unity *(a, b)=1*.

- Definition of Divides: If a and b are integers, we say a divides b, $a|b$, if there is an integer c such that $b=ac$.
- Definition of Modulo: Let m be a positive integer. If a and b are integers, we say that a is congruent to b modulo m, $a=b(mod\ m)$, if $m|(a-b)$ [Ref. 6].

Now, having reviewed the relevant definitions, the Chinese Remainder Theorem can be introduced:

Suppose the moduli $m_1, m_2, \ldots, m_n$ are relatively prime in pairs, that is, $(m_i, m_j)=1$ for all i and j satisfying $1 \le i < j \le n$. Then, the system of congruences,

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\vdots$$
$$x \equiv a_n \pmod{m_n}$$

has a solution. If we *define* $M=m_1 \cdot m_2 \cdots m_n$, and if for $i=1,2,\ldots,n$ b_i is chosen so that $(M/m_i)b_i \equiv 1 \pmod{m_i}$, then x satisfies congruences if and only if [Ref. 7]:

$$x \equiv (M/m_1)b_1a_1+(M/m_2)b_2a_2+\ldots+(M/m_n)b_na_n \pmod{M} \qquad (2.6)$$

In other words, the Theorem states that if an unknown x is divided by m_i and the remainder is a_i, the result can be written using the following notation:

$$x \equiv a_i (\mathrm{mod} m_i) \tag{2.7}$$

where i = 1,2,…,n.

Then, in order to find the solution to the problem, we should find $\{b_i\}$ values by using:

$$(M/m_i) b_i \equiv (\mathrm{mod}\ m_i) \tag{2.8}$$

and putting these values into the Equation 2.6:

$$x = \left(\sum_{i=0}^{n} (M/m_i)\, b_i a_i \right) \mathrm{mod}(M) \tag{2.9}$$

where n is the total number of arrangements, and b_i is an integer.

Let us solve a sample problem to demonstrate the application of this theorem:

Given a_1=3, m_1=7; a_2=2, m_2=10; a_3=5, m_3=11, (notice that 7, 10, and 11 are relatively prime), let us find x.

Since we have earlier stated $M=m_1 \cdot m_2 \cdots m_n$, we can easily find that M=7·10·11=770 and the $\{b_i\}$ can be obtained by trial and error [Ref. 8]:

$$(M/m_1) b_1 = 110 \cdot 3 \equiv 1 \ (\mathrm{mod}\ m_1) \Rightarrow b_1 = 3$$

$$(M/m_2) b_2 = 77 \cdot 3 \equiv 1 \ (\mathrm{mod}\ m_2) \Rightarrow b_2 = 3$$

$$(M/m_3) b_3 = 70 \cdot 3 \equiv 1 \ (\mathrm{mod}\ m_3) \Rightarrow b_3 = 3$$

Then:

$$x = (3 \cdot 110 \cdot 3 + 2 \cdot 77 \cdot 3 + 5 \cdot 70 \cdot 3) \ \mathrm{mod}(770)$$

$$=(990+462+1,050) \bmod(770)$$

$$=2,052 \bmod(770)$$

$$=192$$

C. GENERAL PRODUCT RULE

Suppose a set consists of ordered collections of k elements (k-tuples) and that there are n_1 possible choices for the first element; for each choice of the first element, there are n_2 possible choices of the second element and so on. Finally, for each possible choice of the first k-1 elements, there are n_k choices of the kth element. Then there are $n_1 n_2 \ldots\ldots n_k$ possible k-tuples [Ref. 9].

III. WIDEBAND DIGITAL COMPRESSIVE RECEIVERS

A. INTRODUCTION

Compressive Receivers for SIGINT applications against radar and communications emitters have the following theoretical capabilities [Ref 1]:

- Allows wideband direction finding in a dense signal environment: Due to their ability to separate multiple wideband emitters whose signals overlap in the time domain. The compressive receiver when used in an interferometer provides the only direction finding method which:
  - ♦ Retains amplitude and phase information
  - ♦ Provides 100 percent Probability of Intercept
- Queueing Receiver in Wideband Dense Signal Environment
- Channelized receivers
- Superhetrodyne analysis receivers
- Provides minimal hardware for a Dense Environment Receiver

- Interception of Frequency Hopping Signals
- Interception of Instantaneous Wideband Signals

Although these theoretical advantages of Compressive Receivers have been recognized for many years, their practical implementation has been limited to experimental units in the ELINT frequency band (2 - 18 GHz); and operational units in the COMINT frequency band (2 - 500 MHz) due to limitations in the areas of:

- *The analog implementation*: The state-of-the-art technology in wideband dispersive delay lines limits the overall bandwidth. When analog technology is implemented, the bandwidth is limited to about 1 GHz in practical units. Some experimental work has been attempted to extend their bandwidth, however these units require a cryogenic cooler and suffer from poor resolution.
- *The digital processing of the serial output data*: The state-of-the art technology in digital processing speed limits the performance of Compressive Receivers. The clock speed of the digital output circuitry is directly related to the

bandwidth of the receiver. A 1 GHz bandwidth receiver requires digital circuitry that operates at 1 GHz, while a receiver which operates at 8 GHz would require 8 GHz digital circuitry. The current state-of-the art in this area is in the 1 -2 GHz region.

Schleher has investigated the feasibility of a new type of wideband digital compressive receiver [Ref. 1]. This receiver would cover a bandwidth of 8 GHz, with 100 percent probability of intercept, operating using digital circuitry with clock speeds of 1 GHz. This compressive receiver has the following objectives:

- Increase bandwidths achievable with compressive receivers to the order of 8 GHz
- Replace limiting analog circuitry with digital circuitry
- Provide 100% Probability of Intercept in a single channel
- Allow digital readout circuitry to run at a maximum 1 GHz rate
- Be a precursor to developing a compressive receiver which would cover the full ELINT band of 2-18 GHz

The feasibility of a wideband digital compressive receiver with 8 GHz bandwidth was demonstrated through simulation in 1997 [Ref. 1]. The receiver possesses a zoom feature that allows the output analysis window to run at a 1 GHz rate. This effectively circumvents the problem of running the analysis window at an 8 GHz rate, which is beyond the present capability of digital circuitry.

B. COMPRESSIVE RECEIVER THEORY OF OPERATION

The principle of the basic compressive receiver is shown in Figure 3.1. The incoming signal, $f(t)$, is mixed (multiplied) with a down-chirp waveform so that each Fourier component is converted into a chirp of the same frequency-time slope, but with a starting frequency fixed by that of the original component. The mixed output is then processed in a chirp filter (dispersive delay line) of matched slope, which correlates, or pulse compresses, the constituent chirps. The linear correspondence between frequency and time in the dispersive delay line results in the original Fourier components being translated into time displacements between the resulting pulse-compressed outputs [Ref. 10].

C. STEPPED-FREQUENCY COMPRESSIVE RECEIVERS

With reference to Figure 3.1, the first step in the Compressive Receiver Process is to induce a chirp waveform onto the input signal. This can be accomplished by mixing a fast scanning chirp waveform with the input signal.

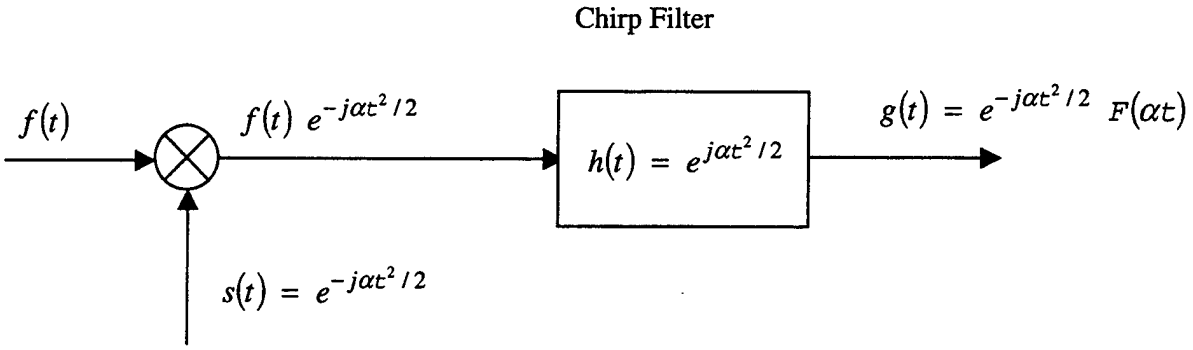

Figure 3.1 - Basic Configuration of Compressive Receiver (After Ref. 1).

The next step is to convert the chirp signal into digital form. This step is a critical one in applying the stepped-frequency methodology. To better understand the significance of the digital conversion, consider the following scenario. Assume that a signal whose center frequency extends over an 8 GHz bandwidth was applied to the compressive receiver. Then the chirp induced onto the signal must have at least 8 GHz bandwidth in order to provide compressive receiver signal action. The net result is that the multiplier has a frequency range that covers a

band 16 GHz wide. To sample this without loss of information requires a Nyquist sample rate in the A/D converter of 32 GHz, which is obviously an impracticably large number.

The stepped-frequency approach segments the chirp waveform into steps where each step covers a portion of the waveform. If eight steps are used, then each step represents a 1 GHz portion of the chirp. Since the total bandwidth is still 8 GHz, no information is lost. Then each chirp segment is translated to baseband where it is processed separately. The Nyquist rate to process each chirp segment is then 2 GHz, rather than 16 GHz.

D. SUB-NYQUIST SAMPLING IN STEPPED-FREQUENCY COMPRESSIVE RECEIVER

Unfortunately, there is still a problem with the stepped-frequency approach. The analog waveform at the output of the stepped-frequency mixer contains both the 8 GHz bandwidth signal and the 1 GHz stepped-frequency chirp carrier, resulting in a total 9 GHz band. This would require an 18 GHz A/D converter for Nyquist rate sampling, which is still an impractical number.

The solution to this problem is to employ sub-Nyquist rate sampling that makes it possible to substantially reduce the bandwidth of the digital processing components.

However, sub-Nyquist rate sampling has the undesirable effect of folding the frequency components of the signal into the fundamental range between $\pm f_s/2$ and, consequently, causing ambiguities. For example, if a 1 GHz sampling rate were used to sample the signal containing the multiplied chirp, then all signals contained in the 8 GHz signal bandwidth would be folded into a 1 GHz wide band. In other words, their frequency is now the measured frequency modulo 1 GHz.

The effect is similar to a zoom lens in a camera as it zooms the sampled signals into a 0-1 GHz range (Figure 3.2). A zoom factor of eight was selected for the current compressive receiver design since this resulted in an A/D converter and digital circuitry design of 1 GHz, which is currently available. The zoom factor results in eight possible frequency ambiguities for each signal located within the 8 GHz band.

Multiple signals that overlap in the time domain would still be resolved; however, the actual center frequency of

the signal would have to be determined by other means. There are several ways that the correct frequency can be determined, but these need to be explored.

The most promising approach appears to be the implementation of a parallel receiver channel using a different zoom factor resulting in a different ambiguity pattern. The correct frequency is then indicated where the patterns coincide for both channels [Ref 1].

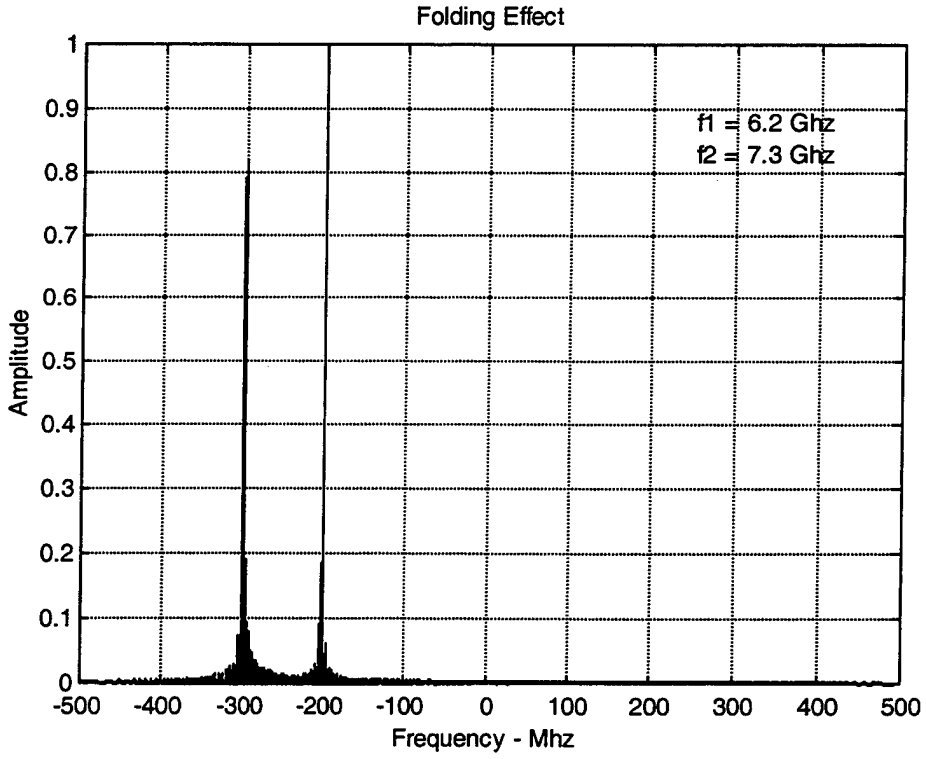

Figure 3.2 - Folding Effect of Sub-Nyquist Sampling In Wideband Compressive Receiver (After Ref. 1).

IV. SOFTWARE DESIGN AND RESULTS

A. INTRODUCTION

As described in Chapter III, sub-Nyquist rate sampling has the undesirable effect of folding the frequency components of the signal into the fundamental range between $\pm f_s/2$ and, consequently, causing ambiguities. It is also shown in Chapter III that if a 1 GHz sampling rate were used to sample the signals in the 0-8 GHz range, then all signals contained in the 8 GHz signal bandwidth would be folded into a 1 GHz wide band (Table 4.1).

	Input Signal Frequency	Sampling Frequency	Output Signal Frequency
1	0.2 GHz	1 GHz	+200 MHz
2	1.4 GHz	1 GHz	+400 MHz
3	2.7 GHz	1 GHz	-300 MHz
4	4.5 GHz	1 GHz	+500 MHz
5	5.5 GHz	1 GHz	+500 MHz
6	6.9 GHz	1 GHz	-100 MHz
7	7.6 GHz	1 GHz	-400 MHz

Table 4.1 - Folding Effect of Sub-Nyquist Sampling on Sampled Signals.

The effect of sub-Nyquist rate sampling can be better understood by contemplating a spinning wheel whose perimeter

is equal to the sampling frequency. Figure 4.1 illustrates this spinning wheel for input signals two and three in Table 4.1. (A minus sign indicates rotation in a counter-clockwise direction.)

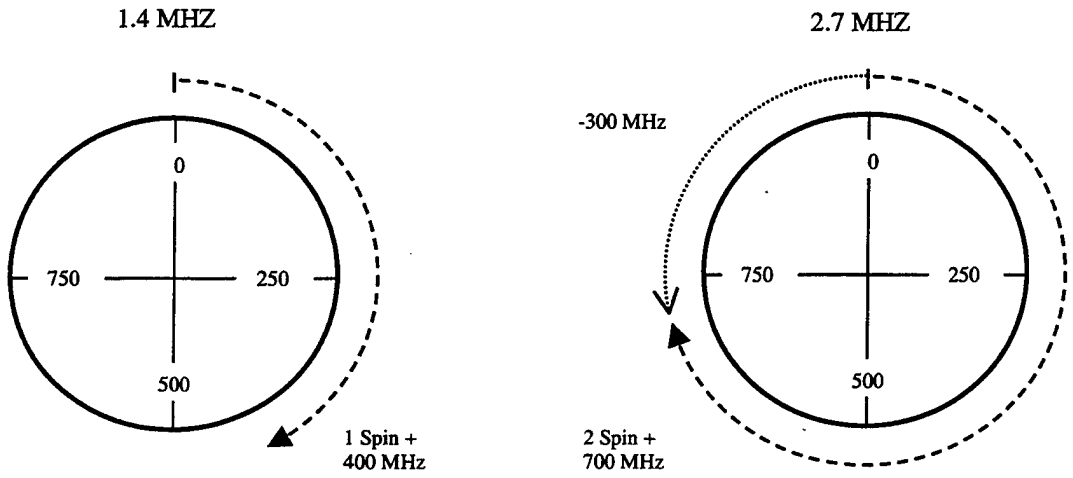

Figure 4.1 - The Effect of Sub-Nyquist Rate Sampling Using a Spinning Wheel.

The results from the spinning wheel will be 400 MHz and -300 MHz for input signals 1.4 GHz and 2.7 GHz, respectively, since it is impossible to keep track of how many times the wheel spins in sub-Nyquist sampling.

Once again, the minus sign indicates rotation in a counter-clockwise direction, so we can say that -300 MHz is equal to 700 MHz in a clockwise direction. Now, in light of this information, we can reconstitute Table 4.1 to see that the input signals will be measured modulo sampling frequency in sub-Nyquist rate sampling (Table 4.2). This

reconstitution is important in order to form an analogy with the Chinese Remainder Theorem (this is presented later).

	Input Signal Frequency	Sampling Frequency	Output Signal Frequency
1	0.2 GHz	1 GHz	+200 MHz
2	1.4 GHz	1 GHz	+400 MHz
3	2.7 GHz	1 GHz	+700 MHz
4	4.5 GHz	1 GHz	+500 MHz
5	5.5 GHz	1 GHz	+500 MHz
6	6.9 GHz	1 GHz	+900 MHz
7	7.6 GHz	1 GHz	+600 MHz

Table 4.2 - Output Signals are Measured Input Signals Modulo Sampling Frequencies.

Notice that although the input signal frequencies are different in rows four and five, the output signal frequencies are the same, meaning that signals 3 and 4 are aliases. Moreover, they are not the only aliases. In fact, if we sample at 1 GHz rate, there are eight aliases in our frequency range of interest, 0-8 GHz, (Table 4.3). As observed in Table 4.3, these aliases occur at frequencies $\pm f_o + m f_s$, where:

f_o = output signal frequency

f_s = sampling frequency

m = 0, 1, 2, … , 7

Hence, there are eight ambiguities for every output signal frequency since each may belong to any of eight input signal frequencies.

	Input Signal Frequency	Sampling Frequency	Output Signal Frequency
1	0.5 GHz	1 GHz	500 MHz
2	1.5 GHz	1 GHz	500 MHz
3	2.5 GHz	1 GHz	500 MHz
4	3.5 GHz	1 GHz	500 MHz
5	4.5 GHz	1 GHz	500 MHz
6	5.5 GHz	1 GHz	500 MHz
7	6.5 GHz	1 GHz	500 MHz
8	7.5 GHz	1 GHz	500 MHz

Table 4.3 - Set of Aliases at 0-8 GHz Range for 1 GHz Sampling Rate.

If we sample at a 1.2 GHz rate, there are seven aliases resulting in seven ambiguities (Table 4.4).

It appears that the Chinese Remainder Theorem can be used to resolve these ambiguities. We can think of the output signal frequencies as remainders modulo the sampling frequency, (a more desirable approach for the purpose of implementing the Chinese Remainder Theorem), and write:

$$f = rem \pmod{f_s} \tag{4.1}$$

where:

f = input signal frequency

rem = remainder which corresponds to output signal frequency

f_s = sampling frequency

	Input Signal Frequency	**Sampling Frequency**	**Output Signal Frequency**
1	0.8 GHz	1.2 GHz	800 MHz
2	2.0 GHz	1.2 GHz	800 MHz
3	3.2 GHz	1.2 GHz	800 MHz
4	4.4 GHz	1.2 GHz	800 MHz
5	5.6 GHz	1.2 GHz	800 MHz
6	6.8 GHz	1.2 GHz	800 MHz
7	8.0 GHz	1.2 GHz	800 MHz

Table 4.4 - Set of Aliases at 0-8 GHz Range for 1.2 GHz Sampling Rate.

In the next step, let us consider that we sample a signal with n different sampling frequencies resulting in n remainders. Then we will have a system of congruencies, i.e.,

$$\begin{aligned} f &= rem1 \pmod{f_1} \\ f &= rem2 \pmod{f_2} \\ &\vdots \\ f &= remn \pmod{f_n} \end{aligned} \qquad (4.2)$$

According to the Chinese Remainder Theorem, this system of congruencies can be solved as long as the sampling frequencies (i.e., the moduli) are relatively prime.

One well known example analogous to our approach is the range ambiguity problem of pulse-Doppler radar [Ref. 11], that can be solved by using multiple pulse repetition frequencies. Another study by Chia shows how the frequency of a signal that is undersampled at two different sampling frequencies can be determined [Ref. 5] for 1 Hz resolution.

In reality, however, we have to work with much higher resolutions. In the next section, we will try to determine the frequency of undersampled signals for higher resolutions.

B. IMPLEMENTATON

An algorithm was developed to solve the frequency ambiguities that occur in our wideband digital compressive receiver design when we use sub-Nyquist sampling for high resolutions, such as 5, 10, or 25 MHz. Note that resolution is defined as the smallest frequency difference for which two simultaneously occuring signals can be resolved in the output window of the compressive receiver.

Then, this algorithm was coded in Matlab® and some parameters were tested to see the effect of:

- Number of signals(f): 1 to 5

- Resolutions (*res*): 5, 10, and 25 MHz
- Number of sampling frequencies: 2 and 3
- Different sampling frequencies (f_s)

The algorithm and testing of the parameters were developed concurrently using a step-by-step procedure, beginning with simple and progressing to complex situations. Problems encountered were resolved by examining the results of the tests.

In this section, the basics of this algorithm are presented along with the problems encountered and their prescribed solutions.

1. Bin Approach

The first difficulty we encounter in the implementation of the Chinese Remainder Theorem is that we have to work with numbers that are too large to efficiently or successfully implement the Theorem, i.e., *6.5e9 modulo 1e9*. In order to overcome this dilemma, we will convert sampling frequencies and remainders to *bins* such that:

$$bin = \frac{frequency}{resolution} \tag{4.3}$$

For example, suppose we sampled a signal at a 1 GHz rate and get 400 MHz as a remainder. Then, for a 25 MHz resolution we will have:

mod_bin=1e9/25e6=40 bins

rem_bin=400 MHz/25 MHz=16 bins

We can repeat the same procedure for another sampling frequency and remainder pair to reconstitute our system of congruencies with more practical numbers. Then, we can easily solve the Equation 4.2 for f. Note that we should multiply the resulting number by the identified resolution since it will also be expressed as bin numbers such that:

$$f(in\ Hz)=(frq\_bin\_)\cdot(res) \tag{4.4}$$

2. Basic Algorithm For One-Signal Case

We will begin an implementation of the Chinese Remainder Theorem with the basic algorithm for the one-signal case shown in Figure 4.2.

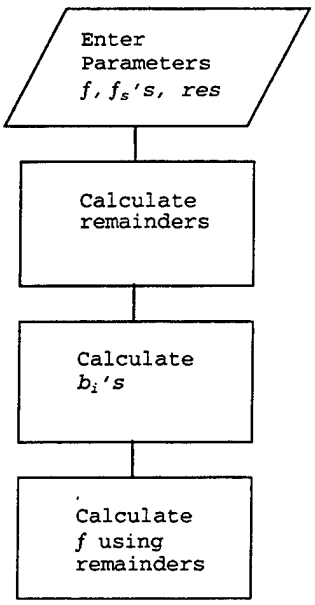

Figure 4.2 - Basic Algorithm for Implementation of Chinese Remainder Theorem.

The software written according to this algorithm can be divided into the following sections:

- Initialization: This section obtains all the parameters required (number of iterations, input signal frequencies, sampling frequencies, resolution).
- Simulation: This section simulates the effect of sub-Nyquist sampling in the Compressive Receiver. It calculates the remainders according to the input signal frequency and sampling frequencies.
- Iteration loop: This section consists of a loop that calculates the b_i values in Equation 2.8, which are necessary to solve for input frequency.
- Calculation of f: In this section, Equation 2.9 is implemented to solve for the input frequency.

3. Modified Algorithm For Multi-Signal Case

The algorithm and the software presented are adequate for a one-signal case. However, there is an additional ambiguity involved in the calculation of f for a multi-signal case. That is, if there is more than one input signal we have to solve for their frequencies.

Let us say that the resulting remainder for a sampled signal is rem*x*_*y*, *x* denoting the sampling signal and *y* denoting the input signal. Then, if two signals are sampled with two sampling frequencies sequentially, there will be

two pairs of remainders, i.e., *rem1_1* and *rem1_2* for the first sampling frequency, and *rem2_1* and *rem2_2* for the second. However, there is no way of knowing how the remainders in each pair will match up with each other.

According to the General Product Rule there are only four ways these remainders can match up:

- *rem1_1 rem2_1*
- *rem1_1 rem2_2*
- *rem1_2 rem2_1*
- *rem1_2 rem2_2*

In this case, it is easy to find the number of possible matches, but it will become more difficult as the number of signals and sampling frequencies increase. For example, there are nine ways of matching for the three signal-case, sixteen for the four signal case, and twenty-five for the five signal case. These numbers would be 27, 64 and 125, respectively, if three sampling frequencies were used instead of two.

Note that we are intuitively referring to the remainders as *rem1_1*, *rem1_2*, etc. Otherwise, it is ambiguous as to which of the two signals is sampled first,i.e. *rem1_(1 or 2)* or *rem2_(1 or 2)*.

Hence, we have to calculate the output frequency for all of the matches and then determine the correct frequencies from them. The various ways to determine the

correct frequencies will be studied after the software has been run and the results have been examined.

As a result of the additional ambiguity caused by multi signals, the basic algorithm presented in Figure 4.2 should be modified for multi-signal cases as seen in Figure 4.3.

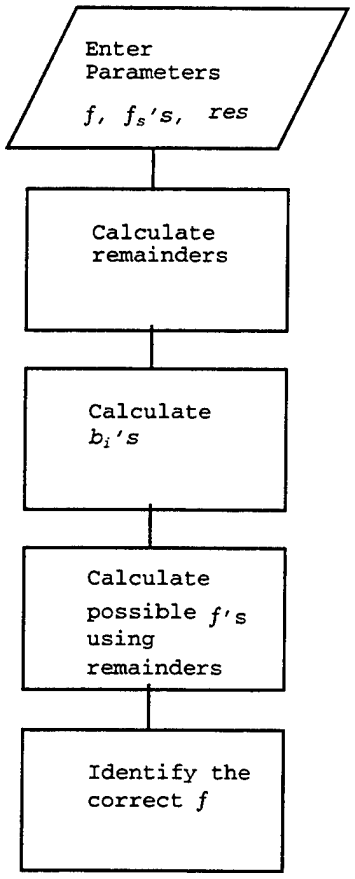

Figure 4.3 - Modified Algorithm for Multi-Signal Cases.

Consequently, the calculation section of the software is modified as follows and an identification section is added:

- Calculation of possible f's: In this section, Equation 2.9 is implemented for all possible matches.
- Identification: The correct outputs are estimated from all possible outputs.

The identification phase of the code is of considerable importance since selecting correct outputs is one of the main challenges targeted in this thesis. While identification is made after carefully examining the results of the software simulations, the main principles of how the identification process works are presented here to facilitate the reader's understanding of the overall procedure.

4. Identification of the Correct Outputs

We have stated that we have to calculate the outputs for all matches, and then determine the correct outputs. Incorrect matches fall into in three categories. (How and why this occurs is described in greater detail in Chapter V.) These categories are defined as:

- Outputs out of 0-8 GHz range
- Outputs the same as the input frequency
- Outputs in the 0-8 GHz range but different than the input frequency

The first type of incorrect outputs does not constitute a problem since we know that we are dealing with signals in

the 0-8 GHz range. This means that in the identification phase, the outputs that do not fall into 0-8 GHz range can be disregarded.

The second type of incorrect output may appear several times before the correct outputs occur, thereby presenting us with the possibility of choosing the same output more than once. To remedy this problem, the identification phase of the code is written so that it selects the first output that appears in the 0-8 GHz range, and then accepts the others as the same signal unless their difference is greater than the resolution.

This is also helpful in dealing with some outputs of the third type, (i.e., the ones whose difference from the input signal is smaller than the resolution). They will also be disregarded since they will be accepted as being the same as the first chosen output. Of course, in this situation, the first chosen output may be the result of an incorrect match; however, the difference between the output of an incorrect match and output of a correct match will be smaller than the resolution.

Other outputs of the second type, (i.e., the ones whose difference with the input signals is greater than the resolution) present the greatest challenge. Since they appear in the 0-8 GHz range and their difference with the

input signal is high enough, they will be mistakenly interpreted as a correct output. One way to alleviate this problem is to detect the parameters where the second type outputs appear the least. This is the main method highlighted in this thesis, and is presented in the next section.

Another way to approach the problem is to use a procedure that distinguishes the incorrect outputs from the correct ones; and subsequently eliminates those incorrect outputs. Fortunately, the Sampling Theorem is helpful in this procedure, as is discussed in the following sub-section.

5. Using the Sampling Theorem for Identification

As described in Chapter II, the Fourier transform of a sampled signal is a periodic replication of the unsampled signal spectrum where each spectral order is separated by the frequency f_s.

If we sample signal x with two different sampling frequencies, it results in *rem1_x* and *rem2_x*. With the information we have about the remainders, we can surmise, (according to the Sampling Theorem) that for the sampling frequency 1, the correct output will show up in the

following set: (note that for multi-signal cases, *x* will have different values, i.e., 1, 2, 3,…,8):

(0*fs1)+rem*1*_*x*
(1*fs1)+rem*1*_*x*
(2*fs1)+rem*1*_*x*
(3*fs1)+rem*1*_*x*
(4*fs1)+rem*1*_*x*
(5*fs1)+rem*1*_*x*
(6*fs1)+rem*1*_*x*
(7*fs1)+rem*1*_*x*
(8*fs1)+rem*1*_*x*

There will be a similar set of possible frequencies for the second sampling frequency:

(0*fs2)+rem*2*_*x*
(1*fs2)+rem*2*_*x*
(2*fs2)+rem*2*_*x*
(3*fs2)+rem*2*_*x*
(4*fs2)+rem*2*_*x*
(5*fs2)+rem*2*_*x*
(6*fs2)+rem*2*_*x*
(7*fs2)+rem*2*_*x*
(8*fs2)+rem*2*_*x*

We know that the correct outputs have to appear in all sets. Now, we can modify the identification phase of the code to check whether the outputs are in these sets, and to disregard them if not. That is, if any of the outputs do

not appear in all of the sets described above, then they will be eliminated, even though they are otherwise correct.

C. EXPERIMENTS WITH TWO SAMPLING FREQUENCIES

Five Matlab programs (Appendix A) were written to implement the algorithms presented earlier for one and multi-signal cases. Ten sets of fifty simulations were made for each different parameter, i.e., number of signals, different sampling frequency pairs, and different resolutions. A Matlab® program (finput.m, Appendix A) was used to generate random inputs in the 0-8 GHz range. This program utilizes the Matlab function *rand* which generates uniformly distributed random numbers in the (0,1) interval.

The resolutions 25, 10, and 5 MHz were chosen for simulations because they are the resolutions that are achievable with our current design.

The sampling frequency pairs are selected to give primarily relative mode bins for each resolution, (i.e., 40-47 and 40-59 for 25 MHz). Additionally, the difference between the second pair is greater than that of the first pair, so that we can observe whether this difference

affects the outcomes. Note that the sampling frequencies are also achievable with current digital technology.

The parameters used and the mean and standard deviation of the number of erroneous results for each set of experiments were presented in Table 4.5 through Table 4.7. The number of erroneous results was presented in Appendix B.

1. 25 MHz Resolution

Mode bin pairs for the chosen sampling frequencies are successively 40-47 and 40-59.

Sampling Frequency 1, fs1	Sampling Frequency 2, fs2	Number of Signals	Mean	St.Dev.
1e9	1.175e9	1	0	0.00
		2	10.6	4.14
		3	26	4.35
		4	37.7	6.73
		5	43.7	5.01
1e9	1.475e9	1	0	0.00
		2	8.9	3.98
		3	24.4	4.25
		4	27.7	5.19
		5	40.8	5.05

Table 4.5 - Mean and Standard Deviation of Number of Erroneous Results with 25 MHz Resolution.

2. 10 MHz Resolution

Mode bin pairs for the chosen sampling frequencies are successively 100-103 and 103-147.

3. 5 MHz Resolution

Mode bin pairs for the chosen sampling frequencies are successively 200-213 and 213-341.

Sampling Frequency 1, fs1	Sampling Frequency 2, fs2	Number of Signals	Mean	St.Dev.
1e9	1.03e9	1	0	0.00
		2	5.5	2.64
		3	13	2.67
		4	24	3.40
		5	30.2	2.78
1.03e9	1.47e9	1	0	0.00
		2	3	1.15
		3	9.2	1.75
		4	16.7	1.83
		5	25.1	5.00

Table 4.6 - Mean and Standard Deviation of Number of Erroneous Results with 10 MHz Resolution.

Sampling Frequency 1, fs1	Sampling Frequency 2, fs2	Number of Signals	Mean	St.Dev.
1e9	1.065e9	1	0	0.00
		2	1.9	1.52
		3	6.8	2.39
		4	12.3	2.58
		5	14.9	5.47
1.065e9	1.705e9	1	0	0.00
		2	1	0.94
		3	3.9	1.37
		4	8.4	2.01
		5	13.5	1.78

Table 4.7 - Mean and Standard Deviation of Number of Erroneous Results with 5 MHz Resolution.

D. EXPERIMENTS WITH THREE SAMPLING FREQUENCIES

The Matlab® programs presented in the previous section were modified in order to use three sampling frequencies instead of two (Appendix A). Ten sets of 50 experiments,

again, were done with the same parameters used with two-sampling frequency experiments to examine whether there is an improvement or not (Table 4.8 through Table 4.9).

The same considerations for resolution and sampling frequencies are taken into account.

1. 25 MHz Resolution

Mode bin triples for the chosen sampling frequencies successively are 40-47-53 and 40-53-67.

Sampling Frequency 1, fs1	Sampling Frequency 2, fs2	Sampling Frequency 3, fs3	Number of Signals	Mean	St.Dev.
1e9	1.175e9	1.325e9	1	0	0.00
			2	0	0.00
			3	1.2	1.03
			4	3.1	2.28
			5	4.1	1.29
1e9	1.265e9	1.435e9	1	0	0.00
			2	0	0.00
			3	0.4	0.52
			4	2.4	0.97
			5	3.4	1.78

Table 4.8 - Mean and Standard Deviation of Number of Erroneous Results with 25 MHz Resolution.

2. 10 MHz Resolution

Mode bin pairs for the chosen sampling frequencies are successively 100-103-107 and 100-127-161.

3. 5 MHz Resolution

Mode bin pairs for the chosen sampling frequencies are successively 200-213-227 and 200-253-287.

Sampling Frequency 1, fs1	Sampling Frequency 2, fs2	Sampling Frequency 3, fs3	Number of Signals	Mean	St.Dev.
1e9	1.03e9	1.07e9	1	0	0.00
			2	0	0.00
			3	0.2	0.42
			4	0	0.00
			5	0.5	0.71
1e9	1.27e9	1.61e9	1	0	0.00
			2	0	0.00
			3	0	0.00
			4	0.1	0.32
			5	0.4	0.70

Table 4.9 - Mean and Standard Deviation of Number of Erroneous Results with 10 MHz Resolution.

Sampling Frequency 1, fs1	Sampling Frequency 2, fs2	Sampling Frequency 3, fs3	Number of Signals	Mean	St.Dev.
1e9	1.065e9	1.0135e9	1	0	0.00
			2	0	0.00
			3	0	0.00
			4	0	0.00
			5	0.1	0.32
1e9	1.265e9	1.435e9	1	0	0.00
			2	0	0.00
			3	0	0.00
			4	0.1	0.32
			5	0.2	0.42

Table 4.10 - Mean and Standard Deviation of Number of Erroneous Results with 5 MHz Resolution.

V. ANALYSIS OF THE RESULTS

After the simulations, the results of the experiments are examined and the following observations made. The examples given to illustrate the observations are for a 25 MHz resolution and a two sampling frequency situation.

First, the same output for slightly different input frequencies is produced. This is due to rounding effects. Since the Chinese Remainder Theorem works only with integers, the bin numbers calculated for remainder frequencies are rounded to the nearest integer. As can be seen on Table 5.1, although the bin numbers for different input frequencies are slightly different before rounding, they are the same after rounding. This characteristic leads to the calculation of exactly the same output for different input frequencies.

Second, it is found that some outputs are slightly different than the inputs (slightly meaning the difference is smaller than the resolution). This is because of the restriction to the integer field when working with the Chinese Remainder Theorem. Due to this restriction, the remainder to frequency mapping is not a continuous

transformation. As a result, the solution for frequency is highly sensitive to the rounding of remainders [Ref.11],

Input frequency	2.34e9	2.345e9	2.3455e9
First remainder	3.4e8	3.45e8	3.455e8
rem1_bin(no rounding)	**13.6**	**13.8**	**13.82**
rem1_bin(with rounding)	**14**	**14**	**14**
Second remainder	1.165e9	1.17e9	1.1705e9
rem2_bin(no rounding)	**46.6**	**46.8**	**46.82**
rem2_bin(with rounding)	**47**	**47**	**47**
f_bin	**94**	**94**	**94**
Output frequency	2.35e9	2.35e9	2.35e9

Table 5.1 - Rounding Effect on Slightly Different Input Frequencies.

causing the output to be different than the input when the remainder bin is rounded to the nearest integer. This effect can be seen in Table 5.2.

Input frequency	2.32e9	2.34e9
First remainder	3.2e8	3.4e8
rem1_bin(no rounding)	**12.8**	**13.6**
rem1_bin(with rounding)	**13**	**14**
Second remainder	1.145e9	1.165e9
rem2_bin(no rounding)	**45.8**	**46.6**
rem2_bin(with rounding)	**46**	**47**
f_bin	**93**	**94**
Output frequency	2.325e9	2.35e9

Table 5.2 - Integer Field Restriction in Chinese Remainder Theorem.

The maximum differences between the inputs and outputs observed for different resolutions can be seen in Table 5.3. As can be observed, the difference never exceeds the resolution, and thus is deemed acceptable.

Resolution (MHz)	25	10	5
Maximum difference between input and output (MHz)	12.5	5	2

Table 5.3 - The Maximum Differences Observed Between Input and Output.

The third observation made is that the intended resolutions are achieved. Despite the integer field and rounding effects mentioned above, different outputs are produced for inputs with the difference equal to the resolution (Table 5.4). This compensates for the undesired effects of the first observation, (i.e., producing the same output for slightly different inputs), such that if the difference between inputs is greater than the resolution, the outputs are different.

Input frequency	**2.200e9**	**2.225e9**	**2.250e9**
First remainder	2e8	2.25e8	2.5e8
Second remainder	1.025e9	1.05e9	1.075e9
f_bin	88	89	90
Output frequency	**2.200e9**	**2.225e9**	**2.250e9**

Table 5.4 - Different Outputs are Produced for Inputs with a Difference Equal to Resolution.

The fourth observation is related to the incorrect matches mentioned in the multi-signal case. They produce outputs in three categories:

- *Outputs out of the 0-8 GHz range.*
- *Outputs that are the same as the input frequency:* If you sample inputs *1e9*, *4e9*, and *8e9* with sampling frequency *1e9*, all the resulting remainder bin numbers will be equal (*bin1_1=bin1_2=bin1_3=0* in this case). Hence, for the input signal 1, not only the correct match bin*1_1-bin2_1-bin3_1*, but also the incorrect matches *bin1_2-bin2_1-bin31* and *bin1_3-bin2_1-bin3_1* will give the correct result, since *bin1_1= bin 1_2= bin 13*.
- *Outputs in the 0-8 GHz range that are different than the input frequency*: Sometimes different input signals result in a very close bin numbers (i.e., *bin1_1*=36 and *bin1_3*=37) when sampled. As a result, the incorrect match *bin1_3-bin2_1-bin3_1* results in a number very close to the correct match, match *bin1_1-bin2_1-bin3_1*. While some of these outputs are very close to the correct inputs, (i.e., their difference is smaller than the resolution), the

others are not as close, (i.e., their difference is greater than the resolution).

The fifth observation concerns how the number of erroneous results changes with the parameters. As seen in Table 5.5, the number of erroneous results increases as: (1) the resolution increases, (2) the number of input signals increases, (3) the number of sampling frequencies decreases, and (4) the difference(s) between sampling frequencies decrease(s). Each of these events will be discussed in greater detail in the following paragraphs.

The first situation occurs because the integer field and rounding effects previously described increase with greater resolution. This is due to the fact that we have to round more to achieve greater resolutions.

The second situation is caused by outputs that are in the 0-8 GHz range, but also different than the input signals previously described. In order to understand how the increase happens, let us consider the four-signal case with three sampling frequencies. There are four bin numbers for each sampling frequency, a total of 12 for all three sampling frequencies i.e.,

Bin1_1	Bin2_1	Bin3_1
Bin1_2	Bin2_2	Bin3_2
Bin1_3	Bin2_3	Bin3_3
Bin1_4	Bin2_4	Bin3_4

These 12 bins result in 4*4*4=64 possible matches (which is eight for the two-signal case and 27 for the three-signal case) as identified in the multi-signal case. Since more bins are matched, the probability of some incorrect outputs occurring through coincidence in the correct frequency range (0-8 GHz) increases. As a result, the number of erroneous results also increases due to the large number of possible signal frequencies.

As the number of sampling frequencies increases, the possibility of getting incorrect outputs occurring in the 0-8 GHz range decreases. In order for the incorrect bins to correctly produce results, they must be close to one of the actual bins, and match up with the corresponding correct bin(s). When there are three sampling frequencies instead of two, the incorrect bins should match up with two correct bins, and so on. Note that there is a trade-off here, since the number of matches also increases with the number of sampling frequencies.

As to the effect of sampling frequencies with greater differences, the sampling frequencies that are very close to each other produce remainders that are also very close to

each other. These remainders, eventually, produce very similar bin numbers. As a result of these incorrect but not highly erroneous bin numbers, we end up with incorrect output frequencies in the correct range. Using sampling frequencies with greater differences avoids this problem.

Number of signals	Means of the Number of Erroneous Results by Using two sampling frequencies					
	25/pair1	25/pair2	10/pair1	10/pair1	5/pair1	5/pair2
1	0	0	0	0	0	0
2	10.6	8.9	5.5	3	1.9	1
3	26	24.4	13	9.2	6.8	3.9
4	37.7	27.7	24	16.7	12.3	8.4
5	43.7	40.8	30.2	25.1	14.9	13.5
Number of signals	**Means of the Number of Erroneous Results by Using three sampling frequencies**					
	25/pair1	**25/pair2**	**10/pair1**	**10/pair1**	**5/pair1**	**5/pair2**
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	1.2	0.4	0.2	0	0	0
4	3.1	2.4	0	0.1	0	0.1
5	4.1	3,4	0.5	0.4	0.1	0.2

Table 5.5 - Means of the Number of Erroneous Results.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

The main objective of this thesis was to resolve the frequency ambiguities that occur in wideband digital compressive receivers using sub-Nyquist sampling as proposed by Schleher [Ref. 1]. For the purposes of analyzing and addressing the aspects of the problem, an algorithm that implements the Chinese Remainder Theorem was developed and presented.

First, it was shown that *output* of sub-Nyquist sampling is congruent with *the sampled signal* modulo *sampling frequency*. Hence, sampling the same signal with multi sampling frequencies forms a simultaneous congruence problem that can be solved using the Chinese Remainder Theorem. Second, an algorithm that implements this approach was developed and coded in Matlab®.

Several experiments with the code were made to see the effects of different parameters on the outcome, i.e., number of signals, different number of sampling frequencies, different sampling frequency pairs and triples, and different resolutions. The algorithm was modified according to the problems encountered and

solutions were provided against these problems. Moreover, some conclusions are drawn according to the results of these experiments.

Conclusions drawn from these experiments suggest that in order to decrease the number of erroneous results that the algorithm produces we should:

- Decrease the resolution
- Increase the number of sampling frequencies
- Increase the difference between sampling frequencies

It is important to emphasize that there are still some aspects of the ambiguity problem in wideband compressive receivers to be analyzed. As a suggestion for future work, it would be interesting to examine the situations for more than five signals occurring concurrently.

Great progress has been achieved in this thesis towards the goal of developing an algorithm to solve the frequency ambiguities. The algorithm has successfully resolved the frequency ambiguiities completely for all resolutions in the 1, and 2-signal cases. The ambiguities have been resolved to a great extent for 10 and 5 MHz resolutions in the 3, 4 and 5-signal cases.

APPENDIX A. MATLAB SOURCE CODES

The following Matlab codes are presented in this Appendix:

FILE NAME	Number of Sampling Frequencies	Number of Input Signals	Modification
crt_2sf_1sgn.m	2	1	No
crt_2sf_2sgn.m	2	2	No
crt_2sf_3sgn.m	2	3	No
crt_2sf_4sgn.m	2	4	No
crt_2sf_5sgn.m	2	5	No
crt_3sf_1sgn.m	3	1	No
crt_3sf_2sgn.m	3	2	No
crt_3sf_3sgn.m	3	3	No
crt_3sf_4sgn.m	3	4	No
crt_3sf_5sgn.m	3	5	No
finput.m	Generates random numbers in the 0-8 GHz range		

Table A.1 - List of Matlab Programs.

```
%crt_2sf_1sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');          %the first
sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');         %the second
sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');
%the resolution that will be used
itr=50;                                                    %how many
debugging tries that will be made
finput
   for n=1:itr
      f_input(n)=finput1;                                  %the
frequency of the first input signal
      mod1=fs1/res;                                       %mod1 is
first sampling frequency in terms of bins
      quot1(n)=fix(f_input(n)/fs1);                       %quotient
      rem1(n)=f_input(n)-quot1(n)*fs1 ;                   %the first
remainder
      bin1(n)=round(mod1*(rem1(n)/fs1)) ;                 %estimated
bin number for remainder1

      mod2=fs2/res;                                       %mod2 is
second sampling frquency in terms of bins
      quot2(n)=fix(f_input(n)/fs2);                       %quotient
      rem2(n)=f_input(n)-quot2(n)*fs2 ;                   %the second
remainder
      bin2(n)=round(mod2*(rem2(n)/fs2));                  %estimated
bin number for remainder2

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2;                              %M is an integer that
is defined in CRT
      b1=0;                                     %integer we have to
find in order to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1;
         b1=b1+1;
      end
      b2=0;                                     %integer we have to
find in order to solve for real frequency
```

```
      while mod((M/mod2)*b2,mod2)~=1;
         b2=b2+1;
      end

      %calculation part: calcuates the input frequency using remainders
      sgn_bin(n)=0;                                  %signal in terms of
bin numbers
      sgn_bin(n)=(M/mod1)*bin1(n)*b1+(M/mod2)*bin2(n)*b2;
      sgn_bin=mod(sgn_bin,M);                        %real frequency in
terms of bin numbers
      sgn_frq=sgn_bin*res ;                          %real frequency in
terms of frequency

      %output
      format short e
      output=[f_input
              sgn_frq]
   end



%crt_2sf_2sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');          %the first
sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');         %the second
sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');
%the resolution that will be used
itr=50;                                                    %how many
debugging tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;                                 %the
frequency of the first input signal
      f_input2(n)=finput2;                                 %the
frequency of the second input signal
      %simulation part:finds the remainders and estimates bin numbers
for them
```

```
      mod1=fs1/res;                                          %mod1 is
first sampling frequency in terms of bins
      quot11(n)=fix(f_input1(n)/fs1);                        %first
quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;                    %first
remainder for sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));                   %first
estimated bin number for sf1

      quot12(n)=fix(f_input2(n)/fs1);                        %second
quotient for sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;                    %second
remainder for sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));                   %second
estimated bin number for sf1

      mod2=fs2/res;                                          %mod1 is
first sampling frequency in terms of bins
      quot21(n)=fix(f_input1(n)/fs2);                        %first
quotient for sf2
      rem21(n)=f_input1(n)-quot21(n)*fs2;                    %first
remainder for sf2
      bin21(n)=round(mod2*(rem21(n)/fs2));                   %first
estimated bin number for sf2

      quot22(n)=fix(f_input2(n)/fs2);                        %second
quotient for sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;                    %second
remainder for sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));                   %second
estimated bin number for sf2

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2;                                    %M is an integer
that is defined in CRT
      b1=0;                                           %integer we have to
find in order to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1
         b1=b1+1;
      end
      b2=0;                                           %integer we have to
find in order to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1
         b2=b2+1;
      end

      %calculation part: calcuates the input frequency using remainders
      sgn_bin(n,4)=0;     %signal in terms of bin numbers
      sgn_bin(n,1)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,2)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,3)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,4)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers
```

```
      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency

      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                 than the former choosen one(s)
      % third, if so assign the "sgn_frq" to the "signal"

      signal1(n)=-1;    %-1 is an "empty flag" indicating no frequency
has been
      signal2(n)=-1;    %choosen for the signal yet

      for m=1:4
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:4
         if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:4
         if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end
```

```
        for m=1:4
          if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
             if signal1(n)==-1
                signal1(n)=sgn_frq(n,m);
             elseif signal2(n)==-1 & ...
                   (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res)
                signal2(n)=sgn_frq(n,m);
             end
          end
       end

       %output
       format short e
       output=[f_input1
               f_input2
               signal1
               signal2]
   end
```

```
%crt_2sf_3sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');          %the first
sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');         %the second
sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');
%the resolution that will be used
itr=50;                                                    %how many
debugging tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;                                 %the
frequency of the first input signal
      f_input2(n)=finput2;                                 %the
frequency of the second input signal
```

```
      f_input3(n)=finput3;                                    %the
frequency of the third input signal
      %simulation part:finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;                                           %mod1
is first sampling frequency in terms of bins
      quot11(n)=fix(f_input1(n)/fs1);                         %first
quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;                     %first
remainder for sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));                    %first
estimated bin number for sf1

      quot12(n)=fix(f_input2(n)/fs1);
%second quotient for sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;
%second remainder for sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));
%second estimated bin number for sf1

      quot13(n)=fix(f_input3(n)/fs1);                         %third
quotient for sf1, signal3
      rem13(n)=f_input3(n)-quot13(n)*fs1;                     %third
remainder for sf1, signal3
      bin13(n)=round(mod1*(rem13(n)/fs1));                    %third
estimated bin number for sf1, signal3

      mod2=fs2/res;                                           %mod2
is second sampling frequency in terms of bins
      quot21(n)=fix(f_input1(n)/fs2);                         %first
quotient for sf2
      rem21(n)=f_input1(n)-quot21(n)*fs2;                     %first
remainder for sf2
      bin21(n)=round(mod2*(rem21(n)/fs2));                    %first
estimated bin number for sf2

      quot22(n)=fix(f_input2(n)/fs2);
%second quotient for sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;
%second remainder for sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));
%second estimated bin number for sf2

      quot23(n)=fix(f_input3(n)/fs2);                         %third
quotient for sf2, signal3
      rem23(n)=f_input3(n)-quot23(n)*fs2;                     %third
remainder for sf2, signal3
      bin23(n)=round(mod2*(rem23(n)/fs2));                    %third
estimated bin number for sf2, signal3

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2;                                    %M is an integer
that is defined in CRT
```

```
      b1=0;                                                %integer we have to
find in order to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1
         b1=b1+1;
      end
      b2=0;                                                %integer we have to
find in order to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1
         b2=b2+1;
      end

      %calculation part: calcuates the input frequency using remainders
      sgn_bin(n,9)=0;          %signal in terms of bin numbers
      sgn_bin(n,1)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,2)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,3)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,4)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,5)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,6)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,7)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,8)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,9)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers
      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency

      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                 than the former choosen one(s)
      % third, if so assign the "sgn_frq" to the "signal".

      signal1(n)=-1;   %-1 is an "empty flag" indicating no frequency
has been
      signal2(n)=-1;   %choosen for the signal yet
      signal3(n)=-1;

      for m=1:9
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:9
         if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem13(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end
```

```
      for m=1:9
         if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem23(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem23(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem23(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem23(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem23(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem23(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem23(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem23(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(8*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem23(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:9
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            if signal1(n)==-1
               signal1(n)=sgn_frq(n,m);
            elseif signal2(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res)
               signal2(n)=sgn_frq(n,m);
            elseif signal3(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                  (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res)
               signal3(n)=sgn_frq(n,m);
            end
         end
      end

      %output
      format short e
      output=[f_input1
              f_input2
              f_input3
              signal1
              signal2
              signal3]
   end
```

```
%crt_2sf_4sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
```

```
fs1=input('Enter the first sampling frequency:');          %the first
sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');         %the second
sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');
%the resolution that will be used
itr=50;                                                    %how many
debugging tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;                                 %the
frequency of the first input signal
      f_input2(n)=finput2;                                 %the
frequency of the second input signal
      f_input3(n)=finput3;                                 %the
frequency of the third input signal
      f_input4(n)=finput4;                                 %the
frequency of the fourth input signal
      %simulation part:finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;
      quot11(n)=fix(f_input1(n)/fs1);          %first quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;      %first remainder for
sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));     %first estimated bin
number for sf1

      quot12(n)=fix(f_input2(n)/fs1);          %second quotient for
sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;      %second remainder for
sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));     %second estimated bin
number for sf1

      quot13(n)=fix(f_input3(n)/fs1);          %third quotient for
sf1, signal3
      rem13(n)=f_input3(n)-quot13(n)*fs1;      %third remainder for
sf1, signal3
      bin13(n)=round(mod1*(rem13(n)/fs1));     %third estimated bin
number for sf1, signal3

      quot14(n)=fix(f_input4(n)/fs1);          %fourth quotient for
sf1, signal4
      rem14(n)=f_input4(n)-quot14(n)*fs1;      %fourth remainder for
sf1, signal4
      bin14(n)=round(mod1*(rem14(n)/fs1));     %fourth estimated bin
number for sf1, signal4

      mod2=fs2/res;
      quot21(n)=fix(f_input1(n)/fs2);        %first quotient for sf2
      rem21(n)=f_input1(n)-quot21(n)*fs2;    %first remainder for sf2
      bin21(n)=round(mod2*(rem21(n)/fs2));   %first estimated bin
number for sf2
```

```
      quot22(n)=fix(f_input2(n)/fs2);          %second quotient for sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;      %second remainder for sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));     %second estimated bin
number for sf2

      quot23(n)=fix(f_input3(n)/fs2);          %third quotient for sf2,
signal3
      rem23(n)=f_input3(n)-quot23(n)*fs2;      %third remainder for sf2,
signal3
      bin23(n)=round(mod2*(rem23(n)/fs2));     %third estimated bin
number for sf2, signal3

      quot24(n)=fix(f_input4(n)/fs2);          %fourth quotient for sf2,
signal4
      rem24(n)=f_input4(n)-quot24(n)*fs2;      %fourth remainder for sf2,
signal4
      bin24(n)=round(mod2*(rem24(n)/fs2));     %fourth estimated bin
number for sf2, signal4

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2;                                    %M is an integer
that is defined in CRT
      b1=0;                                           %integer we have to
find in order to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1
         b1=b1+1;
      end
      b2=0;                                           %integer we have to
find in order to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1
         b2=b2+1;
      end

      %calculation of possible signal frequencies
      sgn_bin(n,16)=0;     %signal in terms of bin numbers
      sgn_bin(n,1) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,2) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,3) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,4) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,5) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,6) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,7) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,8) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,9) =(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,10)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,11)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,12)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,13)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,14)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,15)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,16)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers
      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency
```

```
      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                 than the former choosen one(s)
      % third, if so assign the "sgn_frq" to the "signal".

      signal1(n)=-1;   %-1 is an "empty flag" indicating no frequency
has been
      signal2(n)=-1;   %choosen for the signal yet
      signal3(n)=-1;
      signal4(n)=-1;

      for m=1:16
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:16
         if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(0*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(1*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(2*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(3*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(4*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(0*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(1*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(2*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(3*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(4*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs1)+rem14(1,n)+(res/2))      )
               sgn_frq(n,m)=sgn_frq(n,m);
            else
               sgn_frq(n,m)=-2;
            end
         end

         for m=1:16
            if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem24(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(1*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem24(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
          end
       end

       for m=1:16
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            if signal1(n)==-1
               signal1(n)=sgn_frq(n,m);
          elseif signal2(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res)
               signal2(n)=sgn_frq(n,m);
            elseif signal3(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                  (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res)
               signal3(n)=sgn_frq(n,m);
            elseif signal4(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                  (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res) & ...
                  (sgn_frq(n,m)<=signal3(n)-res |
sgn_frq(n,m)>=signal3(n)+res)
               signal4(n)=sgn_frq(n,m);
            end
         end
      end

      %output
      format short e
      output=[f_input1
              f_input2
              f_input3
              f_input4
```

```
            signal1
            signal2
            signal3
            signal4]
   end



%crt_2sf_5sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');          %the first
sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');         %the second
sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');
%the resolution that will be used
itr=50;                                                    %how many
debugging tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;                                 %the
frequency of the first input signal
      f_input2(n)=finput2;                                 %the
frequency of the second input signal
      f_input3(n)=finput3;                                 %the
frequency of the third input signal
      f_input4(n)=finput4;                                 %the
frequency of the fourth input signal
      f_input5(n)=finput5;                                 %the
frequency of the fifth input signal
      %simulation part:finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;
      quot11(n)=fix(f_input1(n)/fs1);          %first quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;      %first remainder for
sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));     %first estimated bin
number for sf1
```

```
      quot12(n)=fix(f_input2(n)/fs1);          %second quotient for
sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;      %second remainder for
sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));     %second estimated bin
number for sf1

      quot13(n)=fix(f_input3(n)/fs1);          %third quotient for
sf1, signal3
      rem13(n)=f_input3(n)-quot13(n)*fs1;      %third remainder for
sf1, signal3
      bin13(n)=round(mod1*(rem13(n)/fs1));     %third estimated bin
number for sf1, signal3

      quot14(n)=fix(f_input4(n)/fs1);          %fourth quotient for
sf1, signal4
      rem14(n)=f_input4(n)-quot14(n)*fs1;      %fourth remainder for
sf1, signal4
      bin14(n)=round(mod1*(rem14(n)/fs1));     %fourth estimated bin
number for sf1, signal4

      quot15(n)=fix(f_input5(n)/fs1);          %fifth quotient for
sf1, signal5
      rem15(n)=f_input5(n)-quot15(n)*fs1;      %fifth remainder for
sf1, signal5
      bin15(n)=round(mod1*(rem15(n)/fs1));     %fifth estimated bin
number for sf1, signal5

      mod2=fs2/res;
      quot21(n)=fix(f_input1(n)/fs2);          %first quotient for sf2
      rem21(n)=f_input1(n)-quot21(n)*fs2;      %first remainder for sf2
      bin21(n)=round(mod2*(rem21(n)/fs2));     %first estimated bin
number for sf2

      quot22(n)=fix(f_input2(n)/fs2);          %second quotient for sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;      %second remainder for sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));     %second estimated bin
number for sf2

      quot23(n)=fix(f_input3(n)/fs2);          %third quotient for sf2,
signal3
      rem23(n)=f_input3(n)-quot23(n)*fs2;      %third remainder for sf2,
signal3
      bin23(n)=round(mod2*(rem23(n)/fs2));     %third estimated bin
number for sf2, signal3

      quot24(n)=fix(f_input4(n)/fs2);          %fourth quotient for sf2,
signal4
      rem24(n)=f_input4(n)-quot24(n)*fs2;      %fourth remainder for sf2,
signal4
      bin24(n)=round(mod2*(rem24(n)/fs2));     %fourth estimated bin
number for sf2, signal4
```

```
      quot25(n)=fix(f_input5(n)/fs2);          %fourth quotient for sf2,
signal5
      rem25(n)=f_input5(n)-quot25(n)*fs2;      %fourth remainder for sf2,
signal5
      bin25(n)=round(mod2*(rem25(n)/fs2));     %fourth estimated bin
number for sf2, signal5

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2;                                    %M is an integer
that is defined in CRT
      b1=0;                                           %integer we have to
find in order to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1
         b1=b1+1;
      end
      b2=0;                                           %integer we have to
find in order to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1
         b2=b2+1;
      end

      %calculation of possible signal frequencies
      sgn_bin(n,25)=0;      %signal in terms of bin numbers
      sgn_bin(n,1) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,2) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,3) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,4) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,5) =(M/mod1)*bin11(n)*b1+(M/mod2)*bin25(n)*b2;
      sgn_bin(n,6) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,7) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,8) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,9) =(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,10)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin25(n)*b2;
      sgn_bin(n,11)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,12)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,13)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,14)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,15)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin25(n)*b2;
      sgn_bin(n,16)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,17)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,18)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,19)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,20)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2;
      sgn_bin(n,21)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin21(n)*b2;
      sgn_bin(n,22)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin22(n)*b2;
      sgn_bin(n,23)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin23(n)*b2;
      sgn_bin(n,24)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin24(n)*b2;
      sgn_bin(n,25)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin25(n)*b2;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers
      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency

      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
```

```
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                 than the former choosen one(s)
      % third, if so assign the "sgn_frq" to the "signal".

      signal1(n)=-1;   %-1 is an "empty flag" indicating no frequency
has been
      signal2(n)=-1;   %choosen for the signal yet
      signal3(n)=-1;
      signal4(n)=-1;
      signal5(n)=-1;

      for m=1:25
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:25
         if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem15(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(6*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem15(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:25
         if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem23(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(2*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem25(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
```

```
      end

       for m=1:25
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            if signal1(n)==-1
               signal1(n)=sgn_frq(n,m);
            elseif signal2(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res)
               signal2(n)=sgn_frq(n,m);
            elseif signal3(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                  (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res)
               signal3(n)=sgn_frq(n,m);
            elseif signal4(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                  (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res) & ...
                  (sgn_frq(n,m)<=signal3(n)-res |
sgn_frq(n,m)>=signal3(n)+res)
               signal4(n)=sgn_frq(n,m);
            elseif signal5(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                  (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res) & ...
                  (sgn_frq(n,m)<=signal3(n)-res |
sgn_frq(n,m)>=signal3(n)+res) & ...
                  (sgn_frq(n,m)<=signal4(n)-res |
sgn_frq(n,m)>=signal4(n)+res)
                signal5(n)=sgn_frq(n,m);
             end
          end
       end
       %output
       format short e
       output=[f_input1
               f_input2
               f_input3
               f_input4
               f_input5
               signal1
               signal2
               signal3
               signal4
               signal5]
    end
end
```

```
%crt_3sf_1sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');          %the first
sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');         %the second
sampling frequency that will be used
fs3=input('Enter the third sampling frequency:');          %the third
sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');
%the resolution that will be used
itr=50;                                                    %how
many debugging tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;                                 %the
frequency of the first input signal
      %simulation part: finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;                                        %mod1 is
first sampling frequency in terms of bins
      quot1(n)=fix(f_input(n)/fs1);                        %quotient
      rem1(n)=f_input(n)-quot1(n)*fs1 ;                    %the first
remainder
      bin1(n)=round(mod1*(rem1(n)/fs1)) ;                  %estimated
bin number for remainder1

      mod2=fs2/res;                                        %mod2 is
second sampling frquency in terms of bins
      quot2(n)=fix(f_input(n)/fs2);                        %quotient
      rem2(n)=f_input(n)-quot2(n)*fs2 ;                    %the second
remainder
      bin2(n)=round(mod2*(rem2(n)/fs2));                   %estimated
bin number for remainder2

      mod3=fs3/res;                                        %mod3 is
third sampling frquency in terms of bins
      quot3(n)=fix(f_input(n)/fs3);                        %quotient
      rem3(n)=f_input(n)-quot3(n)*fs3;                     %the third
remainder
      bin3(n)=round(mod3*(rem3(n)/fs3));                   %estimated
bin number for remainder3
```

```
      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2*mod3;              %M is an integer that is defined in
CRT
      b1=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1;
         b1=b1+1;
      end
      b2=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1;
         b2=b2+1;
      end
      b3=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod3)*b3,mod3)~=1;
         b3=b3+1;
      end

      %calculation part: calcuates the input frequency using remainders
      sgn_bin(n)=0;                  %signal in terms of bin numbers

   sgn_bin(n)=(M/mod1)*bin1(n)*b1+(M/mod2)*bin2(n)*b2+(M/mod3)*bin3(n)*
b3;
      sgn_bin=mod(sgn_bin,M);        %real frequency in terms of bin
numbers

      %correct some erroneous results occur for 10 MHz resolution
      if sgn_bin(n)>892700.5 & sgn_bin(n)<893500.5
         sgn_bin(n)=sgn_bin(n)-892700.5 ;
      elseif sgn_bin(n)>641999.5 & sgn_bin(n)<642799.5
         sgn_bin(n)=sgn_bin(n)-641999.5 ;
      elseif sgn_bin(n)>460100.5 & sgn_bin(n)<460900.5
         sgn_bin(n)=sgn_bin(n)-460100.5 ;
      elseif sgn_bin(n)>432600.5 & sgn_bin(n)<433400.5
         sgn_bin(n)=sgn_bin(n)-432600.5 ;
      elseif sgn_bin(n)>209399.5 & sgn_bin(n)<210199.5
         sgn_bin(n)=sgn_bin(n)-209399.5 ;
      end

      sgn_frq=sgn_bin*res;           %signal in terms of frequency

      %output
      format short e
      output=[f_input
              sgn_frq]
   end
```

```
%crt_3sf_2sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');          %the first
sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');         %the second
sampling frequency that will be used
fs3=input('Enter the third sampling frequency:');          %the third
sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');
%the resolution that will be used
itr=50;                                                        %how
many debugging tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;                                     %the
frequency of the first input signal
      f_input2(n)=finput2;                                     %the
frequency of the second input signal
      %simulation part:finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;                                           %mod1 is
first sampling frequency in terms of bins
      quot11(n)=fix(f_input1(n)/fs1);                         %first
quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;                     %first
remainder for sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));                    %first
estimated bin number for sf1

      quot12(n)=fix(f_input2(n)/fs1);                         %second
quotient for sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;                     %second
remainder for sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));                    %second
estimated bin number for sf1

      mod2=fs2/res;                                           %mod2 is
second sampling frequency in terms of bins
      quot21(n)=fix(f_input1(n)/fs2);                         %first
quotient for sf2
```

```
      rem21(n)=f_input1(n)-quot21(n)*fs2;                    %first
remainder for sf2
      bin21(n)=round(mod2*(rem21(n)/fs2));                   %first
estimated bin number for sf2

      quot22(n)=fix(f_input2(n)/fs2);                        %second
quotient for sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;                    %second
remainder for sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));                   %second
estimated bin number for sf2

      mod3=fs3/res;                                          %mod3 is
third sampling frequency in terms of bins
      quot31(n)=fix(f_input1(n)/fs3);                        %first
quotient for sf3
      rem31(n)=f_input1(n)-quot31(n)*fs3;                    %first
remainder for sf3
      bin31(n)=round(mod3*(rem31(n)/fs3));                   %first
estimated bin number for sf3

      quot32(n)=fix(f_input2(n)/fs3);                        %second
quotient for sf3
      rem32(n)=f_input2(n)-quot32(n)*fs3;                    %second
remainder for sf3
      bin32(n)=round(mod3*(rem32(n)/fs3));                   %second
estimated bin number for sf3

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2*mod3;              %M is an integer that is defined in
CRT
      b1=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1;
         b1=b1+1;
      end
      b2=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1;
         b2=b2+1;
      end
      b3=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod3)*b3,mod3)~=1;
         b3=b3+1;
      end

      %calculation part: calcuates the input frequency using remainders
      sgn_bin(n,8)=0;     %signal in terms of bin numbers

   sgn_bin(n,1)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin3
1(n)*b3;
```

```
    sgn_bin(n,2)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin3
2(n)*b3;

    sgn_bin(n,3)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin3
1(n)*b3;

    sgn_bin(n,4)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin3
2(n)*b3;

    sgn_bin(n,5)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin3
1(n)*b3;

    sgn_bin(n,6)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin3
2(n)*b3;

    sgn_bin(n,7)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin3
1(n)*b3;

    sgn_bin(n,8)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin3
2(n)*b3;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers

      %correct some erroneous results occur for 10 MHz resolution
      if sgn_bin(n)>892700.5 & sgn_bin(n)<893500.5
         sgn_bin(n)=sgn_bin(n)-892700.5 ;
      elseif sgn_bin(n)>641999.5 & sgn_bin(n)<642799.5
         sgn_bin(n)=sgn_bin(n)-641999.5 ;
      elseif sgn_bin(n)>460100.5 & sgn_bin(n)<460900.5
         sgn_bin(n)=sgn_bin(n)-460100.5 ;
      elseif sgn_bin(n)>432600.5 & sgn_bin(n)<433400.5
         sgn_bin(n)=sgn_bin(n)-432600.5 ;
      elseif sgn_bin(n)>209399.5 & sgn_bin(n)<210199.5
         sgn_bin(n)=sgn_bin(n)-209399.5 ;
      end

      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency

      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                 than the former choosen one(s)
      % third, if so assign the "sgn_frq" to the "signal"

      for m=1:8
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end
```

```
      for m=1:8
         if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2))     )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:8
         if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:8
         if ( (sgn_frq(n,m)>=(0*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem31(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(0*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem32(1,n)+(res/2))     )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      signal1(n)=-1;   %-1 is an "empty flag" indicating no frequency
has been
      signal2(n)=-1;   %choosen for the signal yet

      for m=1:8
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            if signal1(n)==-1
               signal1(n)=sgn_frq(n,m);
            elseif signal2(n)==-1 & ...
               (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res)
            signal2(n)=sgn_frq(n,m);
            end
         end
      end

      %output
      format short e
      output=[f_input1
              f_input2
              signal1
              signal2]
   end



%crt_3sf_3sgn.m
%This program implements the Chinese Remainder Theorem
```

```
%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');                %the
first sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');               %the
second sampling frequency that will be used
fs3=input('Enter the third sampling frequency:');                %the
third sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');               %the
resolution that will be used
itr=50;                                                          %how
many debugging tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;                                       %the
frequency of the first input signal
      f_input2(n)=finput2;                                       %the
frequency of the second input signal
      f_input3(n)=finput3;                                       %the
frequency of the third input signal
      %simulation part:finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;                                              %mod1
is first sampling frequency in terms of bins
      quot11(n)=fix(f_input1(n)/fs1);                            %first
quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;                        %first
remainder for sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));                       %first
estimated bin number for sf1

      quot12(n)=fix(f_input2(n)/fs1);
%second quotient for sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;
%second remainder for sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));
%second estimated bin number for sf1

      quot13(n)=fix(f_input3(n)/fs1);                            %third
quotient for sf1, signal3
      rem13(n)=f_input3(n)-quot13(n)*fs1;                        %third
remainder for sf1, signal3
      bin13(n)=round(mod1*(rem13(n)/fs1));                       %third
estimated bin number for sf1, signal3
```

```
      mod2=fs2/res;                                          %mod2
is second sampling frequency in terms of bins
      quot21(n)=fix(f_input1(n)/fs2);                        %first
quotient for sf2
      rem21(n)=f_input1(n)-quot21(n)*fs2;                    %first
remainder for sf2
      bin21(n)=round(mod2*(rem21(n)/fs2));                   %first
estimated bin number for sf2

      quot22(n)=fix(f_input2(n)/fs2);
%second quotient for sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;
%second remainder for sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));
%second estimated bin number for sf2

      quot23(n)=fix(f_input3(n)/fs2);                        %third
quotient for sf2, signal3
      rem23(n)=f_input3(n)-quot23(n)*fs2;                    %third
remainder for sf2, signal3
      bin23(n)=round(mod2*(rem23(n)/fs2));                   %third
estimated bin number for sf2, signal3

      mod3=fs3/res;                                          %mod3
is third sampling frequency in terms of bins
      quot31(n)=fix(f_input1(n)/fs3);                        %first
quotient for sf3, signal1
      rem31(n)=f_input1(n)-quot31(n)*fs3;                    %first
remainder for sf3, signal1
      bin31(n)=round(mod3*(rem31(n)/fs3));                   %first
estimated bin number for sf3, signal1

      quot32(n)=fix(f_input2(n)/fs3);
%second quotient for sf3, signal2
      rem32(n)=f_input2(n)-quot32(n)*fs3;
%second remainder for sf3, signal2
      bin32(n)=round(mod3*(rem32(n)/fs3));
%second estimated bin number for sf3, signal2

      quot33(n)=fix(f_input3(n)/fs3);                        %third
quotient for sf3, signal3
      rem33(n)=f_input3(n)-quot33(n)*fs3;                    %third
remainder for sf3, signal3
      bin33(n)=round(mod3*(rem33(n)/fs3));                   %third
estimated bin number for sf3, signal3

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2*mod3;              %M is an integer that is defined in
CRT
      b1=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1;
```

```
      b1=b1+1;
   end
   b2=0;                          %integer we have to find in order
to solve for real frequency
   while mod((M/mod2)*b2,mod2)~=1;
      b2=b2+1;
   end
   b3=0;                          %integer we have to find in order
to solve for real frequency
   while mod((M/mod3)*b3,mod3)~=1;
      b3=b3+1;
   end

   %calculation part: calcuates the input frequency using remainders
   sgn_bin(n,27)=0;     %signal in terms of bin numbers
   sgn_bin(n,1)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin31(n)*b3;
   sgn_bin(n,2)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin32(n)*b3;
   sgn_bin(n,3)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin33(n)*b3;
   sgn_bin(n,4)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin31(n)*b3;
   sgn_bin(n,5)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin32(n)*b3;
   sgn_bin(n,6)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin33(n)*b3;
   sgn_bin(n,7)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin31(n)*b3;
   sgn_bin(n,8)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin32(n)*b3;
   sgn_bin(n,9)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin33(n)*b3;

   sgn_bin(n,10)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin
31(n)*b3;

   sgn_bin(n,11)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin
32(n)*b3;

   sgn_bin(n,12)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin
33(n)*b3;

   sgn_bin(n,13)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin
31(n)*b3;

   sgn_bin(n,14)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin
32(n)*b3;

   sgn_bin(n,15)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin
33(n)*b3;

   sgn_bin(n,16)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin
31(n)*b3;
```

```
    sgn_bin(n,17)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin
32(n)*b3;

    sgn_bin(n,18)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin
33(n)*b3;

    sgn_bin(n,19)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin
31(n)*b3;

    sgn_bin(n,20)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin
32(n)*b3;

    sgn_bin(n,21)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin
33(n)*b3;

    sgn_bin(n,22)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin
31(n)*b3;

    sgn_bin(n,23)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin
32(n)*b3;

    sgn_bin(n,24)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin
33(n)*b3;

    sgn_bin(n,25)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin
31(n)*b3;

    sgn_bin(n,26)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin
32(n)*b3;

    sgn_bin(n,27)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin
33(n)*b3;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers

      %correct some erroneous results occur for 10 MHz resolution
      if sgn_bin(n)>892700.5 & sgn_bin(n)<893500.5
         sgn_bin(n)=sgn_bin(n)-892700.5 ;
      elseif sgn_bin(n)>641999.5 & sgn_bin(n)<642799.5
         sgn_bin(n)=sgn_bin(n)-641999.5 ;
      elseif sgn_bin(n)>460100.5 & sgn_bin(n)<460900.5
         sgn_bin(n)=sgn_bin(n)-460100.5 ;
      elseif sgn_bin(n)>432600.5 & sgn_bin(n)<433400.5
         sgn_bin(n)=sgn_bin(n)-432600.5 ;
      elseif sgn_bin(n)>209399.5 & sgn_bin(n)<210199.5
         sgn_bin(n)=sgn_bin(n)-209399.5 ;
      end

      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency

      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
```

```
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                 than the former choosen one(s)
      % third, if so assign the "sgn_frq" to the "signal"

      for m=1:27
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:27
         if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem13(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem13(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(2*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem13(1,n)+(res/2))      )
             sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:27
         if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem23(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

       for m=1:27
         if (  (sgn_frq(n,m)>=(0*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem31(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem32(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(3*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem32(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem33(1,n)+(res/2))     )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      signal1(n)=-1;   %-1 is an "empty flag" indicating no frequency
has been
      signal2(n)=-1;   %choosen for the signal yet
      signal3(n)=-1;

      for m=1:27
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            if signal1(n)==-1
               signal1(n)=sgn_frq(n,m);
            elseif signal2(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res)
               signal2(n)=sgn_frq(n,m);
            elseif signal3(n)==-1 & ...
                  (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                  (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res)
```

```
            signal3(n)=sgn_frq(n,m);
         end
      end
   end

   %output
   format short e
   output=[f_input1
           f_input2
           f_input3
           signal1
           signal2
           signal3]
end
```

```
%crt2_3sf_4sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');
%the first sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');
%the second sampling frequency that will be used
fs3=input('Enter the third sampling frequency:');
%the third sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');                %the
resolution that will be used
itr=50;
                                                    %how many debugging
tries that will be made
finput
   for n=1:itr
      f_input1(n)=finput1;
                  %the frequency of the first input signal
      f_input2(n)=finput2;
                        %the frequency of the second input signal
      f_input3(n)=finput3;
                  %the frequency of the third input signal
      f_input4(n)=finput4;
                  %the frequency of the fourth input signal
```

```
      %simulation part:finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;                              %mod1 is first sampling
frequency in terms of bins
      quot11(n)=fix(f_input1(n)/fs1);            %first quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;        %first remainder for
sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));       %first estimated bin
number for sf1

      quot12(n)=fix(f_input2(n)/fs1);            %second quotient for
sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;        %second remainder for
sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));       %second estimated bin
number for sf1

      quot13(n)=fix(f_input3(n)/fs1);            %third quotient for
sf1, signal3
      rem13(n)=f_input3(n)-quot13(n)*fs1;        %third remainder for
sf1, signal3
      bin13(n)=round(mod1*(rem13(n)/fs1));       %third estimated bin
number for sf1, signal3

      quot14(n)=fix(f_input4(n)/fs1);            %fourth quotient for
sf1, signal4
      rem14(n)=f_input4(n)-quot14(n)*fs1;        %fourth remainder for
sf1, signal4
      bin14(n)=round(mod1*(rem14(n)/fs1));       %fourth estimated bin
number for sf1, signal4

      mod2=fs2/res;                              %mod2 is second
sampling frequency in terms of bins
      quot21(n)=fix(f_input1(n)/fs2);            %first quotient for sf2
      rem21(n)=f_input1(n)-quot21(n)*fs2;        %first remainder for
sf2
      bin21(n)=round(mod2*(rem21(n)/fs2));       %first estimated bin
number for sf2

      quot22(n)=fix(f_input2(n)/fs2);            %second quotient for
sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;        %second remainder for
sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));       %second estimated bin
number for sf2

      quot23(n)=fix(f_input3(n)/fs2);            %third quotient for
sf2, signal3
      rem23(n)=f_input3(n)-quot23(n)*fs2;        %third remainder for
sf2, signal3
      bin23(n)=round(mod2*(rem23(n)/fs2));       %third estimated bin
number for sf2, signal3
```

```
      quot24(n)=fix(f_input4(n)/fs2);            %fourth quotient for
sf2, signal4
      rem24(n)=f_input4(n)-quot24(n)*fs2;        %fourth remainder for
sf2, signal4
      bin24(n)=round(mod2*(rem24(n)/fs2));       %fourth estimated bin
number for sf2, signal4

            mod3=fs3/res;                                %mod3 is third
sampling frequency in terms of bins
            quot31(n)=fix(f_input1(n)/fs3);            %first quotient
for sf3, signal1
            rem31(n)=f_input1(n)-quot31(n)*fs3;        %first remainder
for sf3, signal1
            bin31(n)=round(mod3*(rem31(n)/fs3));       %first estimated
bin number for sf3, signal1

            quot32(n)=fix(f_input2(n)/fs3);            %second quotient
for sf3, signal2
            rem32(n)=f_input2(n)-quot32(n)*fs3;        %second remainder
for sf3, signal2
            bin32(n)=round(mod3*(rem32(n)/fs3));       %second estimated
bin number for sf3, signal2

            quot33(n)=fix(f_input3(n)/fs3);                 %third
quotient for sf3, signal3
            rem33(n)=f_input3(n)-quot33(n)*fs3;        %third remainder
for sf3, signal3
            bin33(n)=round(mod3*(rem33(n)/fs3));       %third estimated
bin number for sf3, signal3

            quot34(n)=fix(f_input4(n)/fs3);            %fourth quotient
for sf3, signal4
            rem34(n)=f_input4(n)-quot34(n)*fs3;        %fourth remainder
for sf3, signal4
            bin34(n)=round(mod3*(rem34(n)/fs3));       %fourth estimated
bin number for sf3, signal4

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2*mod3;                          %M is an integer that
is defined in CRT
      b1=0;                                                %integer we
have to find in order to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1;
            b1=b1+1;
      end
      b2=0;                                                %integer we
have to find in order to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1;
            b2=b2+1;
      end
      b3=0;                                                %integer we
have to find in order to solve for real frequency
      while mod((M/mod3)*b3,mod3)~=1;
```

```
            b3=b3+1;
        end

        %calculation part: calcuates the input frequency using remainders
            sgn_bin(n,64)=0;      %signal in terms of bin numbers
            sgn_bin(n,1)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin31(n)*b3;
            sgn_bin(n,2)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin32(n)*b3;
            sgn_bin(n,3)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin33(n)*b3;
            sgn_bin(n,4)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin34(n)*b3;
            sgn_bin(n,5)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin31(n)*b3;
            sgn_bin(n,6)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin32(n)*b3;
            sgn_bin(n,7)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin33(n)*b3;
            sgn_bin(n,8)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin34(n)*b3;
            sgn_bin(n,9)=
 (M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin31(n)*b3;

       sgn_bin(n,10)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
 bin32(n)*b3;

       sgn_bin(n,11)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
 bin33(n)*b3;

       sgn_bin(n,12)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
 bin34(n)*b3;

       sgn_bin(n,13)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
 bin31(n)*b3;

       sgn_bin(n,14)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
 bin32(n)*b3;

       sgn_bin(n,15)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
 bin33(n)*b3;

       sgn_bin(n,16)=(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
 bin34(n)*b3;

       sgn_bin(n,17)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
 bin31(n)*b3;

       sgn_bin(n,18)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
 bin32(n)*b3;

       sgn_bin(n,19)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
 bin33(n)*b3;
```

```
      sgn_bin(n,20)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,21)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,22)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,23)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,24)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,25)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,26)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,27)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,28)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,29)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,30)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,31)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,32)=(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,33)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,34)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,35)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,36)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,37)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin31(n)*b3;
```

```
      sgn_bin(n,38)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,39)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,40)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,41)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,42)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,43)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,44)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,45)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,46)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,47)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,48)=(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,49)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,50)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,51)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,52)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,53)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,54)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,55)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin33(n)*b3;
```

```
      sgn_bin(n,56)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,57)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,58)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,59)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,60)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*
bin34(n)*b3;

      sgn_bin(n,61)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin31(n)*b3;

      sgn_bin(n,62)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin32(n)*b3;

      sgn_bin(n,63)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin33(n)*b3;

      sgn_bin(n,64)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*
bin34(n)*b3;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers

      %correct some erroneous results occur for 10 MHz resolution
      if sgn_bin(n)>892700.5 & sgn_bin(n)<893500.5
            sgn_bin(n)=sgn_bin(n)-892700.5 ;
            elseif sgn_bin(n)>641999.5 & sgn_bin(n)<642799.5
            sgn_bin(n)=sgn_bin(n)-641999.5 ;
            elseif sgn_bin(n)>460100.5 & sgn_bin(n)<460900.5
            sgn_bin(n)=sgn_bin(n)-460100.5 ;
            elseif sgn_bin(n)>432600.5 & sgn_bin(n)<433400.5
            sgn_bin(n)=sgn_bin(n)-432600.5 ;
            elseif sgn_bin(n)>209399.5 & sgn_bin(n)<210199.5
            sgn_bin(n)=sgn_bin(n)-209399.5 ;
            end

      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency

      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                                  than the former choosen
one(s)
      % third, if so assign the "sgn_frq" to the "signal"
```

```
            for m=1:64
            if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
                  sgn_frq(n,m)=sgn_frq(n,m);
            else
                  sgn_frq(n,m)=-2;
            end
            end

            for m=1:64
            if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
                  (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
                  (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
                  (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(0*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem13(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(1*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem13(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(2*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem13(1,n)+(res/2)) |...
                     (sgn_frq(n,m)>=(3*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem13(1,n)+(res/2)) |...
```

```
                    (sgn_frq(n,m)>=(4*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(4*fs1)+rem13(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs1)+rem13(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs1)+rem13(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs1)+rem13(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs1)+rem13(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs1)+rem13(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(0*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(0*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(1*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(1*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(2*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(2*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(3*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(3*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(4*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(4*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs1)+rem14(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs1)+rem14(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs1)+rem14(1,n)+(res/2))      )
                    sgn_frq(n,m)=sgn_frq(n,m);
              else
                    sgn_frq(n,m)=-2;
              end
              end

              for m=1:64
              if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
```

```
                    (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2)) |...
                          (sgn_frq(n,m)>=(0*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(1*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(2*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(3*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(4*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem23(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(0*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(1*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(2*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(3*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(4*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem24(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem24(1,n)+(res/2))     )
```

```
                sgn_frq(n,m)=sgn_frq(n,m);
            else
                sgn_frq(n,m)=-2;
            end
            end

          for m=1:64
             if (  (sgn_frq(n,m)>=(0*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem31(1,n)+(res/2)) |...
                   (sgn_frq(n,m)>=(1*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem31(1,n)+(res/2)) |...
                   (sgn_frq(n,m)>=(2*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem31(1,n)+(res/2)) |...
                   (sgn_frq(n,m)>=(3*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem31(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(4*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem31(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(5*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem31(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(6*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem31(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(7*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem31(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(8*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem31(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(0*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem32(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(1*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem32(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(2*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem32(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(3*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem32(1,n)+(res/2)) |...
                      (sgn_frq(n,m)>=(4*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem32(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem32(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem32(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem32(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem32(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(0*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(1*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(2*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(3*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(4*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem33(1,n)+(res/2)) |...
```

```
                    (sgn_frq(n,m)>=(5*fs3)+rem33(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs3)+rem33(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs3)+rem33(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs3)+rem33(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs3)+rem33(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(0*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(0*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(1*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(1*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(2*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(2*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(3*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(3*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(4*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(4*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(5*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(5*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(6*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(6*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(7*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(7*fs3)+rem34(1,n)+(res/2)) |...
                    (sgn_frq(n,m)>=(8*fs3)+rem34(1,n)-(res/2) &
 sgn_frq(n,m)<=(8*fs3)+rem34(1,n)+(res/2))     )
                    sgn_frq(n,m)=sgn_frq(n,m);
              else
                    sgn_frq(n,m)=-2;
              end
              end

              signal1(n)=-1;    %-1 is an "empty flag" indicating no
 frequency has been
              signal2(n)=-1;    %choosen for the signal yet
              signal3(n)=-1;
              signal4(n)=-1;

              for m=1:64
              if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
                    if signal1(n)==-1
                          signal1(n)=sgn_frq(n,m);
                    elseif signal2(n)==-1 & ...
                          (sgn_frq(n,m)<=signal1(n)-res |
 sgn_frq(n,m)>=signal1(n)+res)
                          signal2(n)=sgn_frq(n,m);
                    elseif signal3(n)==-1 & ...
                          (sgn_frq(n,m)<=signal1(n)-res |
 sgn_frq(n,m)>=signal1(n)+res) & ...
                          (sgn_frq(n,m)<=signal2(n)-res |
 sgn_frq(n,m)>=signal2(n)+res)
                    signal3(n)=sgn_frq(n,m);
                    elseif signal4(n)==-1 & ...
```

```
                        (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                        (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res) & ...
                        (sgn_frq(n,m)<=signal3(n)-res |
sgn_frq(n,m)>=signal3(n)+res)
                        signal4(n)=sgn_frq(n,m);
                end
            end
            end

            %output
            format short e
            output=[f_input1
                    f_input2
                    f_input3
              f_input4
              signal1
              signal2
              signal3
              signal4]
   end



 %crt_3sf_5sgn.m
%This program implements the Chinese Remainder Theorem

%To solve for f==ai (mod mi) where i=1,2,...,r ("==" indicates
%congruance and mi are pairwise relatively prime,(i.e. their
%greatest common divisor is 1), the Chinese Remainder Theorem
%states that there is a unique solution modulo M=m1*m2*...mr.

%A standard  method of solution is to find integers bi such that
%M/mi*bi=1 (mod mi) where i=1,2,...,r and substitute them into
%equation f=M/m1*a1*bi+M/m2*a2*b2+...+M/mr*ar*br (mod M)


clear all
%initialization part: obtains all the parameters required
fs1=input('Enter the first sampling frequency:');                %the
first sampling frequency that will be used
fs2=input('Enter the second sampling frequency:');               %the
second sampling frequency that will be used
fs3=input('Enter the third sampling frequency:');                %the
third sampling frequency that will be used
res=input('Enter the resolution(25, 10, 5 MHz):');               %the
resolution that will be used
itr=50;                                                          %how
many debugging tries that will be made
finput
   for n=1:itr
```

```
      f_input1(n)=finput1;                                        %the
frequency of the first input signal
      f_input2(n)=finput2;                                        %the
frequency of the second input signal
      f_input3(n)=finput3;                                        %the
frequency of the third input signal
      f_input4(n)=finput4;                                        %the
frequency of the fourth input signal
      f_input5(n)=finput5;                                        %the
frequency of the fifth input signal
      %simulation part:finds the remainders and estimates bin numbers
for them
      mod1=fs1/res;                          %mod1 is first sampling
frequency in terms of bins
      quot11(n)=fix(f_input1(n)/fs1);        %first quotient for sf1
      rem11(n)=f_input1(n)-quot11(n)*fs1;    %first remainder for
sf1
      bin11(n)=round(mod1*(rem11(n)/fs1));   %first estimated bin
number for sf1

      quot12(n)=fix(f_input2(n)/fs1);        %second quotient for
sf1
      rem12(n)=f_input2(n)-quot12(n)*fs1;    %second remainder for
sf1
      bin12(n)=round(mod1*(rem12(n)/fs1));   %second estimated bin
number for sf1

      quot13(n)=fix(f_input3(n)/fs1);        %third quotient for
sf1, signal3
      rem13(n)=f_input3(n)-quot13(n)*fs1;    %third remainder for
sf1, signal3
      bin13(n)=round(mod1*(rem13(n)/fs1));   %third estimated bin
number for sf1, signal3

      quot14(n)=fix(f_input4(n)/fs1);        %fourth quotient for
sf1, signal4
      rem14(n)=f_input4(n)-quot14(n)*fs1;    %fourth remainder for
sf1, signal4
      bin14(n)=round(mod1*(rem14(n)/fs1));   %fourth estimated bin
number for sf1, signal4

      quot15(n)=fix(f_input5(n)/fs1);        %fifth quotient for
sf1, signal5
      rem15(n)=f_input5(n)-quot15(n)*fs1;    %fifth remainder for
sf1, signal5
      bin15(n)=round(mod1*(rem15(n)/fs1));   %fifth estimated bin
number for sf1, signal5

     mod2=fs2/res;                           %mod2 is second
sampling frequency in terms of bins
      quot21(n)=fix(f_input1(n)/fs2);        %first quotient for sf2
      rem21(n)=f_input1(n)-quot21(n)*fs2;    %first remainder for
sf2
```

```
      bin21(n)=round(mod2*(rem21(n)/fs2));      %first estimated bin
number for sf2

      quot22(n)=fix(f_input2(n)/fs2);           %second quotient for
sf2
      rem22(n)=f_input2(n)-quot22(n)*fs2;       %second remainder for
sf2
      bin22(n)=round(mod2*(rem22(n)/fs2));      %second estimated bin
number for sf2

      quot23(n)=fix(f_input3(n)/fs2);           %third quotient for
sf2, signal3
      rem23(n)=f_input3(n)-quot23(n)*fs2;       %third remainder for
sf2, signal3
      bin23(n)=round(mod2*(rem23(n)/fs2));      %third estimated bin
number for sf2, signal3

      quot24(n)=fix(f_input4(n)/fs2);           %fourth quotient for
sf2, signal4
      rem24(n)=f_input4(n)-quot24(n)*fs2;       %fourth remainder for
sf2, signal4
      bin24(n)=round(mod2*(rem24(n)/fs2));      %fourth estimated bin
number for sf2, signal4

      quot25(n)=fix(f_input5(n)/fs2);           %fourth quotient for
sf2, signal5
      rem25(n)=f_input5(n)-quot25(n)*fs2;       %fourth remainder for
sf2, signal5
      bin25(n)=round(mod2*(rem25(n)/fs2));      %fourth estimated bin
number for sf2, signal5

      %sample the signals 1, 2, 3, 4, and 5 with fs3
      mod3=fs3/res;                             %mod2 is second
sampling frequency in terms of bins
      quot31(n)=fix(f_input1(n)/fs3);           %first quotient for
fs3, signal1
      rem31(n)=f_input1(n)-quot31(n)*fs3;       %first remainder for
fs3, signal1
      bin31(n)=round(mod3*(rem31(n)/fs3));      %first estimated bin
number for fs3, signal1

      quot32(n)=fix(f_input2(n)/fs3);           %second quotient for
fs3, signal2
      rem32(n)=f_input2(n)-quot32(n)*fs3;       %second remainder for
fs3, signal2
      bin32(n)=round(mod3*(rem32(n)/fs3));      %second estimated bin
number for fs3, signal2

      quot33(n)=fix(f_input3(n)/fs3);           %third quotient for
fs3, signal3
      rem33(n)=f_input3(n)-quot33(n)*fs3;       %third remainder for
fs3, signal3
      bin33(n)=round(mod3*(rem33(n)/fs3));      %third estimated bin
number for fs3, signal3
```

```
      quot34(n)=fix(f_input4(n)/fs3);          %fourth quotient for
fs3, signal4
      rem34(n)=f_input4(n)-quot34(n)*fs3;      %fourth remainder for
fs3, signal4
      bin34(n)=round(mod3*(rem34(n)/fs3));     %fourth estimated bin
number for fs3, signal4

      quot35(n)=fix(f_input5(n)/fs3);          %fourth quotient for
fs3, signal5
      rem35(n)=f_input5(n)-quot35(n)*fs3;      %fourth remainder for
fs3, signal5
      bin35(n)=round(mod3*(rem35(n)/fs3));     %fourth estimated bin
number for fs3, signal5

      %iteration part: calculates bi's, which are necessary to find
"sgn_bin"
      M=mod1*mod2*mod3;              %M is an integer that is defined in
CRT
      b1=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod1)*b1,mod1)~=1;
         b1=b1+1;
      end
      b2=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod2)*b2,mod2)~=1;
         b2=b2+1;
      end
      b3=0;                          %integer we have to find in order
to solve for real frequency
      while mod((M/mod3)*b3,mod3)~=1;
         b3=b3+1;
      end

      %calculation part: calcuates the input frequency using remainders
      sgn_bin(n,125)=0;     %signal in terms of bin numbers
      sgn_bin(n,1)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,2)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,3)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,4)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,5)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,6)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,7)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,8)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin33(n)*b3;
```

```
      sgn_bin(n,9)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,10)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,11)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,12)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,13)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,14)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,15)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,16)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,17)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,18)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,19)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,20)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,21)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,22)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,23)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,24)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,25)=
(M/mod1)*bin11(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,26)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,27)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,28)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,29)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,30)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,31)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,32)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,33)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,34)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,35)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin35(n)*b3;
```

```
      sgn_bin(n,36)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,37)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,38)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,39)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,40)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,41)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,42)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,43)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,44)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,45)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,46)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,47)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,48)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,49)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,50)=
(M/mod1)*bin12(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,51)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,52)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,53)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,54)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,55)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,56)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,57)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,58)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,59)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,60)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,61)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,62)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin32(n)*b3;
```

```
      sgn_bin(n,63)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,64)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,65)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,66)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,67)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,68)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,69)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,70)=
(M/mod1)*bin13(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,71)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,72)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,73)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,74)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,75)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,76)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,77)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,78)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,79)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,80)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,81)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,82)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,83)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,84)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,85)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,86)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,87)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,88)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,89)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin34(n)*b3;
```

```
      sgn_bin(n,90)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,91)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,92)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,93)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,94)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin34(n)*b3;
      sgn_bin(n,95)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin35(n)*b3;
      sgn_bin(n,96)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin31(n)*b3;
      sgn_bin(n,97)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin32(n)*b3;
      sgn_bin(n,98)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin33(n)*b3;
      sgn_bin(n,99)=
(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bin34(n)*b3;

   sgn_bin(n,100)=(M/mod1)*bin14(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bi
n35(n)*b3;

   sgn_bin(n,101)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bi
n31(n)*b3;

   sgn_bin(n,102)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bi
n32(n)*b3;

   sgn_bin(n,103)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bi
n33(n)*b3;

   sgn_bin(n,104)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bi
n34(n)*b3;

   sgn_bin(n,105)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin21(n)*b2+(M/mod3)*bi
n35(n)*b3;

   sgn_bin(n,106)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bi
n31(n)*b3;

   sgn_bin(n,107)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bi
n32(n)*b3;

   sgn_bin(n,108)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bi
n33(n)*b3;

   sgn_bin(n,109)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bi
n34(n)*b3;

   sgn_bin(n,110)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin22(n)*b2+(M/mod3)*bi
n35(n)*b3;
```

```
    sgn_bin(n,111)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bi
n31(n)*b3;

    sgn_bin(n,112)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bi
n32(n)*b3;

    sgn_bin(n,113)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bi
n33(n)*b3;

    sgn_bin(n,114)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bi
n34(n)*b3;

    sgn_bin(n,115)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin23(n)*b2+(M/mod3)*bi
n35(n)*b3;

    sgn_bin(n,116)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bi
n31(n)*b3;

    sgn_bin(n,117)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bi
n32(n)*b3;

    sgn_bin(n,118)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bi
n33(n)*b3;

    sgn_bin(n,119)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bi
n34(n)*b3;

    sgn_bin(n,120)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin24(n)*b2+(M/mod3)*bi
n35(n)*b3;

    sgn_bin(n,121)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bi
n31(n)*b3;

    sgn_bin(n,122)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bi
n32(n)*b3;

    sgn_bin(n,123)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bi
n33(n)*b3;

    sgn_bin(n,124)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bi
n34(n)*b3;

    sgn_bin(n,125)=(M/mod1)*bin15(n)*b1+(M/mod2)*bin25(n)*b2+(M/mod3)*bi
n35(n)*b3;
      sgn_bin=mod(sgn_bin,M);  %real frequency in terms of bin numbers

      %correct some erroneous results occur for 10 MHz resolution
      if sgn_bin(n)>892700.5 & sgn_bin(n)<893500.5
         sgn_bin(n)=sgn_bin(n)-892700.5 ;
      elseif sgn_bin(n)>641999.5 & sgn_bin(n)<642799.5
         sgn_bin(n)=sgn_bin(n)-641999.5 ;
      elseif sgn_bin(n)>460100.5 & sgn_bin(n)<460900.5
         sgn_bin(n)=sgn_bin(n)-460100.5 ;
```

```
      elseif sgn_bin(n)>432600.5 & sgn_bin(n)<433400.5
         sgn_bin(n)=sgn_bin(n)-432600.5 ;
      elseif sgn_bin(n)>209399.5 & sgn_bin(n)<210199.5
         sgn_bin(n)=sgn_bin(n)-209399.5 ;
      end

      sgn_frq=sgn_bin*res;     %real frequency in terms of frequency

      % estimation part: estimates which "sgn_frq" ies are correct
according to the following steps:
      % first, check whether the "sgn_frq" is in the correct range (1-8
GHz)
      % second, check 1)whether the "signal" is empty and,
      %               2)if so, whether the "sgn_frq" is different
      %                 than the former choosen one(s)
      % third, if so assign the "sgn_frq" to the "signal"

      for m=1:125
         if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:125
         if (  (sgn_frq(n,m)>=(0*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(5*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(6*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(7*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(8*fs1)+rem11(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem11(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(0*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(1*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(2*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(3*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem12(1,n)+(res/2)) |...
               (sgn_frq(n,m)>=(4*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem12(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(5*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem12(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem12(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem13(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem13(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem14(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem14(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs1)+rem15(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(6*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs1)+rem15(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs1)+rem15(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs1)+rem15(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
      end

      for m=1:125
         if (  (sgn_frq(n,m)>=(0*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem21(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem21(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem22(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem22(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem23(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(2*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem23(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem23(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem24(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem24(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs2)+rem25(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs2)+rem25(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs2)+rem25(1,n)+(res/2))      )
            sgn_frq(n,m)=sgn_frq(n,m);
         else
            sgn_frq(n,m)=-2;
         end
```

```
      end

      for m=1:125
         if ( (sgn_frq(n,m)>=(0*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(1*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(2*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(3*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(4*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(5*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(6*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(7*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(8*fs3)+rem31(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem31(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(0*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(1*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(2*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(3*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(4*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(5*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(6*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(7*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(8*fs3)+rem32(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem32(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(0*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem33(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(1*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem33(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(2*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem33(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(3*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem33(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(4*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem33(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(5*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem33(1,n)+(res/2)) |...
              (sgn_frq(n,m)>=(6*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem33(1,n)+(res/2)) |...
```

```
                (sgn_frq(n,m)>=(7*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem33(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem33(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem34(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem34(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(0*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(0*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(1*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(1*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(2*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(2*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(3*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(3*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(4*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(4*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(5*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(5*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(6*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(6*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(7*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(7*fs3)+rem35(1,n)+(res/2)) |...
                (sgn_frq(n,m)>=(8*fs3)+rem35(1,n)-(res/2) &
sgn_frq(n,m)<=(8*fs3)+rem35(1,n)+(res/2))      )
                sgn_frq(n,m)=sgn_frq(n,m);
            else
                sgn_frq(n,m)=-2;
            end
        end

        signal1(n)=-1;    %-1 is an "empty flag" indicating no frequency
has been
        signal2(n)=-1;    %choosen for the signal yet
        signal3(n)=-1;
        signal4(n)=-1;
        signal5(n)=-1;

        for m=1:125
```

```
          if sgn_frq(n,m)>=0 & sgn_frq(n,m)<=8e9
             if signal1(n)==-1
                signal1(n)=sgn_frq(n,m);
             elseif signal2(n)==-1 & ...
                   (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res)
                signal2(n)=sgn_frq(n,m);
             elseif signal3(n)==-1 & ...
                   (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                   (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res)
                signal3(n)=sgn_frq(n,m);
             elseif signal4(n)==-1 & ...
                   (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                   (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res) & ...
                   (sgn_frq(n,m)<=signal3(n)-res |
sgn_frq(n,m)>=signal3(n)+res)
                signal4(n)=sgn_frq(n,m);
            elseif signal5(n)==-1 & ...
                   (sgn_frq(n,m)<=signal1(n)-res |
sgn_frq(n,m)>=signal1(n)+res) & ...
                   (sgn_frq(n,m)<=signal2(n)-res |
sgn_frq(n,m)>=signal2(n)+res) & ...
                   (sgn_frq(n,m)<=signal3(n)-res |
sgn_frq(n,m)>=signal3(n)+res) &...
                   (sgn_frq(n,m)<=signal4(n)-res |
sgn_frq(n,m)>=signal4(n)+res)
                signal5(n)=sgn_frq(n,m);
            end
         end
      end

       %output
       format short e
       output=[f_input1
               f_input2
               f_input3
               f_input4
               f_input5
               signal1
               signal2
               signal3
               signal4
               signal5]
   end


%finput.m generates randon numbers in the 0-8 GHz range
%using matlab function "rand"
```

```
rand('state',sum(100*clock))  %resets the generator to a different
state each time
y=rand(5,100);                %generates a 5x500 matrix of random
numbers
y=y*1e10;

l=1;                          %eliminates the elements greater than 8e9
for k=1:500
   if y(k)<=8e9
      x(l)=y(k);
      l=l+1;
   end
end

finput1=x(1:50);              %generates five fifty-element matrices
finput2=x(51:100);
finput3=x(101:150);
finput4=x(151:200);
finput5=x(201:250);
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. THE NUMBER OF ERRONEOUS RESULTS

Resolution (MHz)	Sampling Frequency Pair	Number of Signals	Set									
			1	2	3	4	5	6	7	8	9	10
25	First	One	0	0	0	0	0	0	0	0	0	0
25	First	Two	5	11	16	12	16	9	8	5	9	15
25	First	Three	15	30	27	26	29	26	29	27	28	23
25	First	Four	21	36	44	37	36	43	42	35	41	42
25	First	Five	30	43	45	46	46	45	44	48	44	46
25	Second	One	0	0	0	0	0	0	0	0	0	0
25	Second	Two	0	7	9	7	11	13	10	7	11	14
25	Second	Three	15	24	23	25	25	21	27	25	30	29
25	Second	Four	17	22	32	31	29	32	31	23	29	31
25	Second	Five	27	44	45	42	41	43	41	43	41	41
10	First	One	0	0	0	0	0	0	0	0	0	0
10	First	Two	8	1	6	5	5	7	6	10	5	2
10	First	Three	13	15	8	9	13	14	16	16	13	13
10	First	Four	19	26	21	28	26	26	22	26	19	27
10	First	Five	28	32	33	35	29	27	28	31	27	32
10	Second	One	0	0	0	0	0	0	0	0	0	0
10	Second	Two	1	2	3	2	4	5	4	3	3	3
10	Second	Three	6	11	10	11	9	8	9	10	7	11
10	Second	Four	16	14	16	18	15	17	20	18	15	18
10	Second	Five	17	22	20	21	24	28	31	31	30	27
5	First	One	0	0	0	0	0	0	0	0	0	0
5	First	Two	2	5	3	1	0	3	2	2	0	1
5	First	Three	6	6	8	5	10	8	6	7	2	10
5	First	Four	6	13	12	11	13	12	13	14	16	13
5	First	Five	12	20	16	19	17	2	16	21	13	13
5	Second	One	0	0	0	0	0	0	0	0	0	0
5	Second	Two	0	2	1	1	0	0	1	1	3	1
5	Second	Three	4	3	2	5	2	4	5	5	6	3
5	Second	Four	9	6	5	7	8	11	11	10	9	8
5	Second	Five	11	14	15	12	13	13	12	15	17	13

Table B.1 - The Number of Erroneous Results with Two Sampling Frequencies.

esolution (MHz)	Sampling Frequency Triple	Number of Signals	Set									
			1	2	3	4	5	6	7	8	9	10
25	First	One	0	0	0	0	0	0	0	0	0	0
25	First	Two	0	0	0	0	0	0	0	0	0	0
25	First	Three	0	0	2	2	2	3	1	1	0	1
25	First	Four	5	7	1	1	2	3	3	3	6	0
25	First	Five	3	3	4	3	3	5	6	3	6	5
25	Second	One	0	0	0	0	0	0	0	0	0	0
25	Second	Two	0	0	0	0	0	0	0	0	0	0
25	Second	Three	0	1	0	0	0	1	0	1	0	1
25	Second	Four	2	2	3	1	4	2	1	3	3	3
25	Second	Five	3	6	5	3	3	5	1	2	1	5
10	First	One	0	0	0	0	0	0	0	0	0	0
10	First	Two	0	0	0	0	0	0	0	0	0	0
10	First	Three	0	1	0	0	0	0	1	0	0	0
10	First	Four	0	0	0	0	0	0	0	0	0	0
10	First	Five	0	1	0	1	0	2	1	0	0	0
10	Second	One	0	0	0	0	0	0	0	0	0	0
10	Second	Two	0	0	0	0	0	0	0	0	0	0
10	Second	Three	0	0	0	0	0	0	0	0	0	0
10	Second	Four	0	0	0	0	0	0	0	0	0	1
10	Second	Five	1	0	0	0	0	1	0	0	0	2
5	First	One	0	0	0	0	0	0	0	0	0	0
5	First	Two	0	0	0	0	0	0	0	0	0	0
5	First	Three	0	0	0	0	0	0	0	0	0	0
5	First	Four	0	0	0	0	0	0	0	0	0	0
5	First	Five	0	0	0	0	1	0	0	0	0	0
5	Second	One	0	0	0	0	0	0	0	0	0	0
5	Second	Two	0	0	0	0	0	0	0	0	0	0
5	Second	Three	0	0	0	0	0	0	0	0	0	0
5	Second	Four	0	1	0	0	0	0	0	0	0	0
5	Second	Five	0	0	0	1	0	0	0	1	0	0

Table B.2 - The Number of Erroneous Results with Three Sampling Frequencies.

LIST OF REFERENCES

1. Schleher, D. Curtis, *Wideband Digital Compressive Receivers*, Naval Postgraduate School, CA, December, 1998.
2. James H. McClellan, Ronald W. Schafer, and Mark A. Yoder, *DSP First*, Prentice Hall, NJ, 1998.
3. Schleher, D. Curtis, *EO3402 Signals and Noise Course Notes*, Naval Postgraduate School, 1998.
4. *EA-181 Digital Signal processing Lecture Notes*, National Cryptologic School Science and Technology, National Security Agency Central Security Service, 1995.
5. Eng S. Chia, *Single-Frequency Measurements using Undersampling Methods*, MS Thesis, Naval Postgraduate School, March, 1998.
6. Rosen, Kenneth H., *Elementary Number Theory and its Applications*, Addison-Wesley Pub. Co., 1984.
7. A. M. Kirch, *Elementary Number Theory - A Computer Approach*, Intext Educational Publishers, New York, 1974.

8. Tsui, James, *Digital Techniques for Wideband Receivers,* Artech House, Inc. MA, 1995.

9. Jay L. Devore, *Probability and Statistics for Engineering and the Sciences,* Duxbury Press, San Luis Obispo, 1995.

10. Schleher, D. Curtis, *Introduction to Electronic Warfare,* Artech House, Norwood, MA, 1986.

11. Schleher, D. Curtis, *MTI and Pulsed Doppler Radar,* Artech House, Norwood, MA, 1991.

12. William S. McCormick, James B. Y. Tsui, and Vernon L. Bakke, "A Noise Insensitive Solution to an Ambiguity Problem in Spectral Estimation," *IEEE Trans. Aerospace and Electronic Systems,* Vol. 25, no. 5, pp. 729-732, Sep 1989.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Professor Curtis D. Schleher 1
   Code IW/Sc
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Professor David Jenn 1
   Code EC/Jn
   Naval Postgraduate School
   Monterey, CA 93943-5101

5. Professor Phillip Pace 1
   Code EC/Pc
   Naval Postgraduate School
   Monterey, CA 93943-5101

6. Professor Dan Boger 1
   Chairman, Code IW
   Naval Postgraduate School
   Monterey, CA 93943-5101

7. Genelkurmay Baskanligi 1
   Personel Baskanligi
   Bakanliklar
   Ankara, TURKEY

8. Kara Kuvvetleri Komutanligi 1
   Personel Daire Baskanligi
   Bakanliklar
   Ankara, TURKEY

9. Kara Kuvvetleri Komutanligi 1
Kutuphanesi
Bakanliklar
Ankara, TURKEY

10. Kara Harp Okulu 2
Kutuphanesi
Dikmen
Ankara, TURKEY

11. Oktay Ceylan 2
MEBS Okulu ve Egitim Merkez Komutanligi
Ogretim Baskanligi
Mamak
Ankara, TURKEY